

# AccuRev

Streamlining Change for Software Development



# Using AccuRev



## **Streamlining Change for Software Development**

# **Using AccuRev**

This document includes four books from the AccuRev documentation set.  
The full documentation set is available in PDF format at:

**<http://www.accurev.com>**

For technical support, contact AccuRev Customer Support:

**[support@accurev.com](mailto:support@accurev.com)**

**800-383-8170 (U.S. and Canada)  
+1 781-861-8700 (worldwide)**

# Using AccuRev

## June, 2006

Copyright © AccuRev, Inc. 1995–2006  
ALL RIGHTS RESERVED

**TimeSafe** and **AccuRev** are registered trademarks of AccuRev, Inc.

**AccuBridge**, **AccuReplica**, **AccuWork**, and **StreamBrowser** are trademarks of AccuRev, Inc.

All other trade names, trademarks and service marks in this document are the property of their respective owners.

# Guide to the AccuRev Documentation Set

This document, *Using AccuRev*, contains four books from the online AccuRev documentation set:

- *AccuRev User's Guide (GUI Edition)* (white pages, starting on page 1): Describes the AccuRev Graphical User Interface components— the File Browser, StreamBrowser, History Browser, Version Browser, Stream Version Browser, Change Palette, and Diff, Merge and Patch tools — and how to use them to perform development and configuration tasks.
- *AccuWork Issue Management Manual* (blue pages, starting on page 167): Describes AccuRev's issue management facility, including how it works with issues databases and change packages.
- *AccuRev Administrator's Guide* (ivory pages, starting on page 239): Describes how to manage the AccuRev system, including its server and data repository.
- *AccuRev Integrations Manual* (yellow pages, starting on page 299): Describes how to integrate AccuRev with systems like the Eclipse IDE, Microsoft Source Code Control, and the Sun Java Studio.

The complete documentation set also includes these books:

- *AccuRev Concepts Manual*: Describes the main concepts and the facilities of the AccuRev software configuration management system.
- *AccuRev User's Guide (CLI)*: Describes the AccuRev command line interface, and provides a reference for all user commands.
- *AccuRev Technical Notes*: Covers a series of specific topics, including using triggers, resolving namespace issues, and setting up multi-server environments.

All AccuRev documentation is available on the company web site: **<http://www.accurev.com>**.





# Table of Contents

<b>Guide to the AccuRev Documentation Set .....</b>	<b>iii</b>
<b>Mechanics of the AccuRev GUI .....</b>	<b>1</b>
Getting Started .....	1
The Multiple-Tab Display .....	1
Cloning Tabs, 2	
Refreshing Tabs, 3	
Keyboard Accelerators .....	3
Dialog Boxes for Selecting or Specifying a Filename .....	3
Working with Tables .....	4
Adjusting the Widths and Order of Columns, 4	
Sorting the Rows of a Table, 5	
Using the Keyboard to Navigate through a Table, 6	
User Preferences .....	6
‘General’ Page, 7	
‘Diff/Merge’ Page, 8	
‘Version Browser’ Page, 8	
‘Stream Browser’ Page, 9	
Executing AccuRev Commands .....	9
Selecting Objects, 9	
Selecting a Command, 9	
Main Menu and Toolbar .....	10
Main Menu, 10	
File Menu, 10	
Edit Menu, 12	
Actions Menu, 13	
View Menu, 13	
Issues Menu, 13	
Tools Menu, 13	
Admin Menu, 14	
Help Menu, 15	
Main Toolbar, 15	
Overall Status Indicators .....	16
The GUI Tools .....	16
<b>Day-to-Day Usage of AccuRev .....</b>	<b>19</b>
The AccuRev Usage Model .....	19
Change and Synchronization: The Four Basic Commands .....	20
Keep: Preserving Changes in Your Private Workspace, 20	
The Fine Print, 21	
Promote: Making Your Private Changes Public, 22	
Streams, 22	
Promotion and Parallel Development, 23	
Active and Inactive Files, 23	
The Fine Print, 24	
Update: Incorporating Others’ Changes into Your Workspace, 24	
The Fine Print, 25	

Merge: When Changes Would Collide ..., 25	
The Fine Print, 27	
Learning More about AccuRev.....	28
<b>Sample Development Workflow .....</b>	<b>29</b>
Creating a Workspace .....	29
Navigating a Workspace .....	32
Editing Source Files.....	34
Checkpointing — Saving Private Versions.....	35
Concurrent Development — Incorporating Other Developers’ Work.....	35
Making Your Changes Public .....	36
Concurrent Development — When Streams Collide.....	36
Getting in Touch With Your Past .....	36
The Version Browser, 38	
Comparing Versions, 39	
Advanced Operations.....	40
Migrating Changes from Stream to Stream, 40	
<b>The File Browser .....</b>	<b>43</b>
Alternatives to the File Browser .....	44
Opening a File Browser Tab .....	44
Leaving a File Browser Tab, 44	
File Browser Layout .....	45
Folders Pane, 45	
Searches Pane, 45	
Details Pane, 46	
AccuRev File Statuses .....	48
Controlling the Display of External Objects, 50	
Controlling the Determination of (modified) Status, 50	
Working in the Details Pane .....	50
Common Usage Scenarios, 51	
Editing a File’s Contents, 51	
Renaming or Moving a File, 53	
Following Through by Promoting the Changes, 54	
Following Through by Undoing the Changes, 54	
Changing a Directory, 55	
Merging Your Changes with Someone Else’s Changes, 56	
Deleting a File — Accidentally or Temporarily, 57	
Deleting a File — Intentionally and Permanently, 59	
File Browser Command Reference, 60	
Working in the Folders Pane .....	66
Working in Include/Exclude Mode .....	67
How Include/Exclude Mode Changes the Folders and Details Panes, 67	
Adding Rules, 69	
Example: Excluding a Directory, 69	
Example 1: Simulating a Sparse Workspace, 71	
Example 2: Excluding a Subtree, 72	
Removing Rules, 72	
Leaving Include/Exclude Mode, 73	
Working in the Searches Pane.....	73

Search Criteria, 74	
Controlling the Display of Element Names, 76	
Optimizations in Searches, 76	
Operating on Files Selected by a Search, 76	
Displaying the Contents of a Dynamic Stream or Snapshot .....	77
Updating the Workspace.....	78
Kinds of Changes Involved in an Update, 79	
Deciding Which Files to Update, 79	
Performing the Update, 80	
Recording the Update, 81	
Performance Optimizations .....	81
Timestamp Optimization, 82	
Validity of the Optimization, 84	
Pathname Optimization, 84	
Appendix: File Statuses and Searches .....	85

## **The StreamBrowser .....91**

Opening a StreamBrowser Tab.....	91
Graphical StreamBrowser Display .....	91
Controlling the Display of Streams, Snapshots, and Workspaces, 93	
Zooming In on a Subhierarchy, 94	
Ensuring Performance Improvements, 95	
Manipulating the Stream Hierarchy .....	95
Rearranging Streams and Workspaces, 95	
Reconfiguring a Stream, 96	
Creating New Streams, Snapshots, and Workspaces.....	97
Exploring the Contents of Streams .....	98
Displaying and Working with the Default Group, 98	
Promoting the Entire Default Group, 99	
Streams and Change Packages, 99	
Change Packages “In” Streams, 100	
The ‘Show Issues’ Command.....	101
Working in the Issues Pane of the Stream Issues Tab, 103	
Working in the Change Package Contents Pane of the Stream Issues Tab, 104	
Comparing the Contents of Streams .....	105
Comparing Streams Using Change Packages, 108	
Propagating Versions through the Stream Hierarchy .....	109
Propagating All of a Stream’s Changes, 109	
Propagating Selected Changes from a Stream, 110	
Additional Operations on Streams, Snapshots, and Workspaces .....	110
Stream History .....	112
Tabular StreamBrowser Display.....	114

## **The AccuRev Diff, Merge, and Patch Tools .....115**

Enabling the Diff and Merge Tools .....	115
Invoking the Diff Tool.....	116
Using the Diff Tool.....	117
Difference in Pathname, 117	
Comparison of Binary Files, 117	
Comparison of Text Files, 117	



Navigating the Differences, 119	
Searching for Text, 120	
Editing a File Using the Diff Tool, 120	
When to Use the Merge Tool .....	122
The Merge Algorithm .....	122
A Non-Conflicting Content Change, 123	
A Conflicting Content Change, 123	
Changes at the Namespace Level, 124	
Invoking the Merge Tool.....	124
Resolving a Namespace-Level Conflict .....	125
Viewing and Resolving Content-Level Conflicts .....	125
Navigating the Display, 127	
Searching for Text, 128	
Resolving Conflicts, 128	
The ‘Revert All’ and ‘Use Only’ Buttons, 129	
Manual Editing, 129	
Searching for Text, 130	
Joining Change Sections, 130	
Finishing a Merge Session, 130	
Merging HTML Files, 131	
Merging Binary Files, 131	
Performing a Patch Instead of a Merge.....	132
Patch Example 1, 133	
Patch Example 2, 133	
Indicating a Patch and its Set of Versions, 134	
 <b>The History Browser .....</b>	 <b>135</b>
Invoking the History Browser.....	135
The History Browser Display .....	136
The Summary Pane, 137	
Operations on Transactions in the Summary Pane, 139	
The Comment Pane, 140	
The Versions Pane, 140	
Operations on Versions in the Versions Pane, 141	
Quick Switch: History / Version / Stream Version Browsers .....	142
 <b>The Version Browser: Ancestry Tracking .....</b>	 <b>143</b>
Invoking the Version Browser .....	143
The Version Browser Display .....	143
User Preferences .....	144
Ancestry Relationships .....	145
Direct Ancestor — Modification of an Existing Version, 145	
Alias — Virtual Version Ancestry, 146	
Merge — Merging of Two Versions, 147	
Closest Common Ancestor, 148	
Patch — Selective Merging of Two Versions, 148	
Operations on Versions .....	149
Quick Switch: History / Version / Stream Version Browsers .....	150

<b>The Stream Version Browser .....</b>	<b>151</b>
Invoking the Stream Version Browser.....	151
The Stream Version Browser Display .....	151
Working in the Stream Version Browser .....	151
Quick Switch: History / Version / Stream Version Browsers .....	153
 <b>Using the Change Palette .....</b>	 <b>155</b>
Sending Selected Versions to the Change Palette.....	157
Letting AccuRev Select the Versions to be Sent, 158	
Populating the Change Palette from the History Browser, 159	
Working in the Change Palette .....	161
Specifying the Destination Stream, 161	
Merging the Source and Destination Versions (If Necessary), 162	
Selecting a Workspace for Performing Merges, 162	
Performing the Merges, 163	
Performing the Promotions, 164	
CLI Commands Related to the Change Palette .....	164
 <b>Working with Issue Records .....</b>	 <b>167</b>
Accessing a Particular Issues Database .....	167
Creating a New Issue Record .....	167
Creating Attachments to an Issue Record, 169	
Canceling Creation of a New Issue Record, 171	
Issue Records and Transactions — the Issue History Page, 172	
The server_dispatch_post Trigger, 172	
Viewing and Modifying an Existing Issue Record .....	172
Printing an Individual Issue Record .....	173
Using AccuWork Queries .....	174
Typical Workflow, 176	
Query List Pane, 176	
Query Results Pane, 178	
The Edit Query Window: Creating and Revising Queries, 179	
Naming a New Query / Renaming an Existing Query, 180	
Creating a Simple Clause, 180	
Creating a Compound Clause, 183	
Closing the Edit Query Window, 185	
Working with a Results Table, 185	
Selecting Columns to Appear in the Results Table, 185	
Working with the Records Listed in a Results Table, 186	
Printing Query Results .....	187
 <b>Designing Issues Databases and Edit Forms:</b>	
<b>    The Schema Editor .....</b>	<b>189</b>
Invoking the Schema Editor .....	189
Saving Changes to the Schema.....	190
Defining Database Fields .....	190
Adding and Removing Fields from the Database Schema, 191	
Integrating Configuration Management and Issue Management:	
the ‘affectedFiles’ Field and Change Packages, 192	
Data Types, 193	

Defining Multiple-Choice Lists, 194	
Designing an Edit Form .....	194
Form Layout Operations, 196	
Defining Edit Form Validations .....	198
Initializing Field Values in a New Issue Record .....	200
Conditional Validations .....	200
Specifying the Condition, 201	
Specifying the Actions, 202	
Setting a Field Value, 203	
Revising the Choices for a “choose” Field, 204	
Requiring a Value to be Entered in a Field, 205	
Setting Permissions on All or Part of the Issue Record, 205	
Requiring Change Set Entries, 206	
Revising and Removing Validations and Actions, 206	
The Change Packages Tab .....	207

## **Change Packages and Integrations between Configuration Management and Issue Management .....209**

Structure of a Change Package .....	209
Creating Change Package Entries .....	211
Complex Change Package Entries .....	211
Operations on Change Package Entries .....	213
Updating Change Package Entries .....	214
A Little Bit of Notation, 214	
Combining Two Change Package Entries, 214	
Viewing Stream Contents/Differences in Terms of Change Packages .....	217
Formatting a Change Package Table, 217	
Promote by Change Package, 218	
Viewing a Transaction in Terms of Change Packages .....	218
“Locking” a Change Package .....	219
Integrations Between Configuration Management and Issue Management .....	220
Change-Package-Level Integration, 220	
Enabling the Integration, 220	
Triggering the Integration, 221	
Transaction-Level Integration, 223	
Enabling the Integration, 223	
Triggering the Integration, 224	
Implementation and Customization of the Transaction-Level Integration, 224	
If Both Integrations are Enabled, 225	

## **AccuWork Command-Line Interface .....227**

Overview .....	227
AccuWork CLI Operations .....	228
Determining the Field-ID / Field-Name Correspondence, 228	
Selecting Issue Records with a Query .....	230
More Complex Queries, 230	
Special Field Types, 232	
Creating a New Issue Record .....	233
Modifying an Existing Issue Record .....	234
Using ‘modifyIssue’ to Create New Issue Records, 235	

Interface to the Change Package Facility .....	236
Adding Entries to a Change Package, 236	
Removing Entries from a Change Package, 237	
Listing the Contents of a Change Package, 237	
Listing Transactions that Affected Change Packages, 238	
<b>The AccuRev Repository .....</b>	<b>239</b>
Repository Access Permissions .....	239
READ ME NOW: Assuring the Integrity of the AccuRev Repository .....	239
Backing Up the Repository .....	241
Restoring the Repository .....	242
Archiving Portions of the Repository .....	242
Moving a Workspace or Reference Tree .....	242
Moving a Depot .....	243
Removing a Depot .....	243
A Word of Caution on Windows Zip Utilities .....	243
Storage Layout .....	243
<b>The AccuRev Server .....</b>	<b>245</b>
User Identity of the Server Process .....	245
Unix Systems Only: Administrative User Identities, 245	
Starting the AccuRev Server .....	246
Running the Server Automatically at Operating System Startup, 246	
Starting the Server Manually, 246	
Server Configuration File .....	246
Unix Systems Only: Controlling the Server's User Identity, 247	
Server Watchdog .....	247
Server Logging .....	248
Watchdog Logging, 248	
Controlling Server Operation .....	248
Unix: 'acserverctl' Utility, 248	
Windows: 'Services' Console, 250	
Server-Control Files, 250	
Open Filehandle Limits and the AccuRev Server .....	251
Changing the Per-Process Open File Descriptor Limit, 252	
Linux, 252	
Solaris, 253	
HP-UX, 253	
<b>Archiving of Version Container Files .....</b>	<b>255</b>
The 'archive' Command .....	256
Determining Which Versions to Archive, 256	
Archiving the Versions, 256	
The 'reclaim' Command .....	257
Attempts to Access Archived Versions .....	258
Using 'hist' to Research Previous 'archive' Commands .....	258
Restoring Archived Versions — The 'unarchive' Command .....	258

<b>Replication of the AccuRev Repository .....</b>	<b>261</b>
Master and Replica .....	261
AccuRev Licensing in a Replication Environment.....	263
Installation Procedure: Assumptions .....	263
Setting Up the Master Server.....	263
Setting Up the Replica Server.....	264
Install AccuRev, 264	
Revise the Server Configuration File, 264	
Start the AccuRev Server Process, 265	
Synchronize the Site Slice, 265	
Configure the Replica Server to Include the Desired Depots, 265	
Using the Same Host as Both Master Server and Replica Server .....	266
Setting Up a Client Machine to Use a Replica Server .....	266
Using a Replica Server .....	266
The Update Command, 267	
Creating New Depots.....	267
Adding and Removing Depots from a Replica Repository .....	267
Synchronizing a Replica Manually .....	268
On-Demand Downloading of a Version's Storage File, 268	
Automating Replica Synchronization .....	269
Synchronization Security .....	269
The replica_site.xml File .....	270
 <b>Moving the AccuRev Server and Repository to Another Machine .....</b>	 <b>271</b>
Procedure for Moving the Repository .....	271
On the Destination Machine .....	271
On the Source Machine.....	272
On the Destination Machine .....	273
 <b>AccuRev Triggers .....</b>	 <b>275</b>
Pre-Operation Triggers .....	275
Client-Side Triggers, 275	
Server-Side Triggers, 275	
Post-Operation Triggers.....	276
Trigger for Transaction-Level Integration between Configuration Management and Issue Management.....	276
Preparing to Use an AccuRev-Provided Trigger Script.....	277
Enabling a Trigger .....	278
pre-create-trig, pre-keep-trig, pre-promote-trig, server-post-promote-trig, 278	
server_admin_trig, 278	
server_preop_trig, 278	
server_dispatch_post, 279	
Notes on Triggers in Multiple-Platform Environments, 279	
The Trigger Parameters File .....	279
Format of the “pre-create-trig” Trigger Parameters File, 280	
Overwriting the Trigger Parameters File, 281	
Format of the “pre-keep-trig” Trigger Parameters File, 282	
Format of the “pre-promote-trig” Trigger Parameters File, 283	
Overwriting the Trigger Parameters File, 284	

Format of the “server-post-promote-trig” Trigger Parameters File, 285	
Format of the “server_preop_trig” Trigger Parameters File, 285	
Format of the “server_admin_trig” Trigger Parameters File, 286	
Format of the “server_dispatch_post” Trigger Parameters File, 287	
Encoding of Element Lists, 288	
Encoding of Command Comments, 288	
Trigger Script Contents, 289	
Trigger Script Exit Status, 289	
File Handling by Trigger Scripts, 289	
Trigger Script Execution and User Identities .....	290
‘Administrative Users’ in Trigger Scripts, 290	
The Trigger Log File .....	290
<b>The ‘maintain’ Utility .....</b>	<b>293</b>
‘maintain’ Command Reference.....	293
Backup/Restore of the AccuRev Repository .....	295
Removing a Depot from the AccuRev Repository .....	296
Before You Begin, 296	
Depot Removal Procedure, 296	
<b>Using AccuRev with Eclipse .....</b>	<b>299</b>
Installing the AccuRev Plug-in .....	299
If You’re Upgrading ..., 300	
The AccuRev Usage Model.....	300
Establishing Your Identity .....	302
Creating an Eclipse Project for Your AccuRev Data .....	302
Advanced “Pick and Choose” Techniques, 304	
Working in the Navigator View .....	305
Executing an AccuRev Action, 305	
Output from AccuRev Actions.....	306
AccuRev Console View, 306	
AccuRev Search View, 306	
Searches in Projects that Include Only Part of an AccuRev Workspace, 307	
Persistence of Data in the AccuRev Views, 307	
AccuRev Actions Summary .....	308
Add to AccuRev Depot, 309	
Keep, 309	
Anchor, 309	
Promote, 309	
Keep and Promote, 309	
Defunct, 310	
Revert to Backed Version, 310	
Revert to Most Recent Version, 310	
Diff Against Backed Version, 310	
Diff Against Most Recent Version, 310	
Update AccuRev Workspace, 310	
Update AccuRev Workspace Preview, 311	
AccuRev Workspace Information, 311	
AccuRev Searches, 311	
Searches on Parts of a Workspace, 313	
Non-Team Commands that Invoke AccuRev Commands.....	314



Promote-Based Integrations with Issue Management .....	315
--	-----

## **Using AccuRev with SCC-Compliant Applications .....317**

Installing the Integration.....	317
Turning Off the Integration, 317	
Using a Workspace .....	318
Establishing Your Identity .....	318
SCC Commands and AccuRev Commands.....	318
AccuRev Command Output, 319	
SCC Command Summary.....	319
Add to Source Control, 319	
Check Out, 320	
Working in an Exclusive File Locking Workspace, 321	
Check In, 321	
Undo Check Out, 321	
Get Latest Version, 322	
Show Differences / Compare Versions, 323	
Show History, 324	
Save As, 324	
Remove, 325	
Additional AccuRev-Related Commands.....	325
Run AccuRev, 326	
Plug-In Options, 327	
Synchronize Time, 328	
Update, 328	
Update Preview, 328	
Workspace Information, 328	
Populate Missing Files, 329	
Merge Overlapping Files, 329	
Keep Modified Files, 329	
Promote Kept Files, 329	
Searches / Show, 329	
AccuRev Properties, 329	
Refresh Status, 329	
Promote-Based Integrations with Issue Management .....	330
Notes on Application Usage .....	330
File Status Icons, 331	
Files that Should Not Be Placed under Source Control, 331	

## **Using AccuRev with Sun Java Studio .....333**

Installing the JAR File .....	333
Enabling the Integration.....	333
Disabling the Integration, 333	
The AccuRev Usage Model.....	334
Establishing Your Identity .....	335
Using a Workspace .....	335
AccuRev Command Output.....	336
Command Summary .....	337
Keep, 337	
Promote, 337	
Anchor, 337	

Add to Depot, 338  
Revert to Backed, 338  
Revert to Recent, 338  
Diff, 338  
Update, 338



# Mechanics of the AccuRev GUI

This chapter provides an overview of the AccuRev graphical user interface (GUI), from a “how to turn the knobs and push the buttons” perspective. It also includes descriptions of the main command menu and the main toolbar, which are always available. (The GUI includes a number of tools, such as the File Browser and the Version Browser, with their own commands and toolbars.)


Before continuing, make sure that:

- You have installed the AccuRev client software on your computer.
- The AccuRev server software has been installed on a computer to which you have a network connection. You can run the client and server software on the same machine; this is the typical setup for a product evaluation.

We also recommend that you first read the *AccuRev Concepts Manual*. This can greatly accelerate your learning, because AccuRev works differently than CM systems with a branches-and-labels architecture.

## Getting Started

You can start the AccuRev GUI from the desktop (Windows only) or from a command shell:

- On the Windows desktop, double-click the  **AccuRev** icon. There is also an **AccuRev** entry in the **Start > Programs** menu.
- In a command shell, use the **acgui** command:  

```
> acgui                (Windows)
> acgui &              (Unix)
```

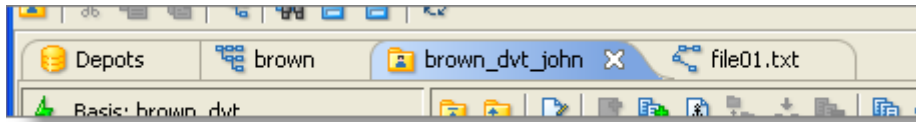
The way the GUI window initially appears varies. It might be blank, or it might open one or more tabs, displaying your work from the preceding GUI session. The AccuRev GUI reopens:

- File Browser tabs, each displaying the contents of an individual workspace or stream.
- StreamBrowser tabs, each displaying the hierarchy of streams, snapshots, and workspaces for a particular depot.
- Queries tabs, each showing the set of AccuWork queries you’ve defined for a particular issues database (and possibly the results of one of those queries).

## The Multiple-Tab Display

The AccuRev GUI window uses multiple tabs (or “windows within a window”) to enable you to switch quickly among several activities. For example, you might wish to switch among:

- Viewing the contents of your main workspace, using the **File Browser**.



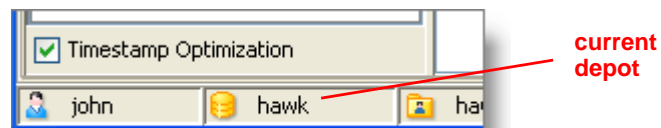
- Browsing through all the versions of a particular file.
- Viewing a list of all workspaces (or just the workspaces that belong to you).
- Viewing the contents of one or your colleague's workspaces.

At any time, you can “clean up”, by closing one or more of the tabs. Right-click on a tab control to display its context menu, then select **Close**. If the AccuRev Look And Feel is enabled (**Tools > Preferences**), an “X” icon appears on the tab control itself. Left-click this icon to close the tab.



Typically, the AccuRev repository is organized into multiple depots. Each depot stores the complete history of one particular directory tree. For example, there might be separate depots for the development, testing, and documentation groups.

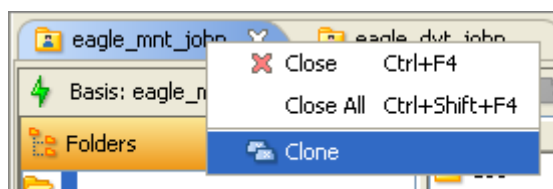
Most GUI tabs display the data from one particular depot. When you're using a particular tab, the associated depot is termed the current depot. Its name is displayed at the bottom of the overall GUI window.



## Cloning Tabs

The AccuRev GUI assumes that you want to avoid having multiple tabs with the same contents. So, for example, if there's already a File Browser tab on workspace **hawk\_mnt\_john** and you execute an **Open Workspace** command on the same workspace, the GUI simply brings the existing tab to the front.

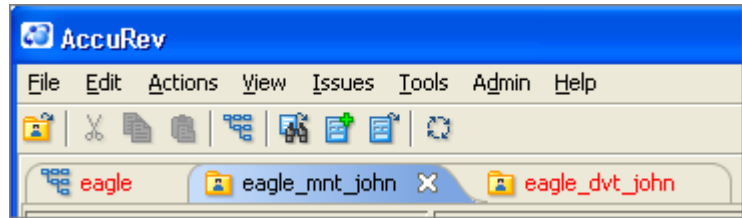
If you do want to have two or more tabs showing the same data structure, use the **Clone** command. This enables you, for example, to work in two different parts of a large stream hierarchy (two StreamBrowser tabs), or to work in two different folders of a workspace (two File Browser tabs).



Note: The History Browser, Version Browser, and Stream Version Browser are not part of this scheme.

## Refreshing Tabs

The data displayed on GUI tab can become out of date as a result of your work on other GUI tabs, your work using the AccuRev CLI, and/or other users' work. AccuRev displays a tab title in red if it determines that the tab's data may be out of date.



The command **View > Refresh** (or function key **F5**) updates a tab's data. You can configure the GUI to refresh tab data automatically whenever you switch tabs: invoke the command **Tools > Preferences**, and clear the **Require Manual Refresh** checkbox.

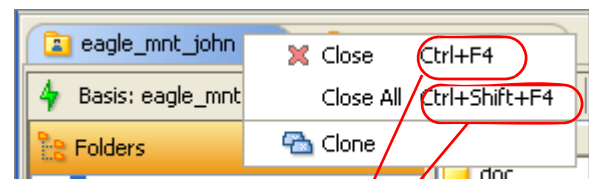
## Keyboard Accelerators

Like all graphical interfaces, the AccuRev GUI includes many menus and dialog boxes. You can access the main menu without having to use the mouse, by using the **Alt** key. For example, pressing **Alt-F** opens the File submenu of the GUI's main menu. You can make submenu choices with additional **Alt** key sequences.

In a dialog box, you can "press" buttons using **Alt** key sequences, such as **Alt-C** for "Cancel". The **Esc** key also performs a "Cancel" operation in any dialog box.

In some dialog boxes, you don't even need to use the **Alt** key when invoking a keyboard accelerator: In a "Yes/No" dialog box, you can type **Y** or **N** (either uppercase or lowercase). In a message or confirmation box (no input field), you can indicate "Ok" by typing **O** (either uppercase or lowercase), or by pressing **Enter**.

Many commands can be invoked with a keyboard accelerator, without opening any menu at all. The accelerators for such commands are listed on the menu itself.



keyboard accelerators for menu commands

## Dialog Boxes for Selecting or Specifying a Filename

In many situations, the AccuRev GUI displays a dialog box, in which you specify a new or existing file on your hard disk. In each context, AccuRev remembers the folder (directory) you specified most recently. This location is used as initial path next time that dialog box appears. If a context has no such "remembered" folder, then your home directory is used.



The dialog box includes a **New Folder** button, enabling you to store a file at a pathname that does not currently exist.

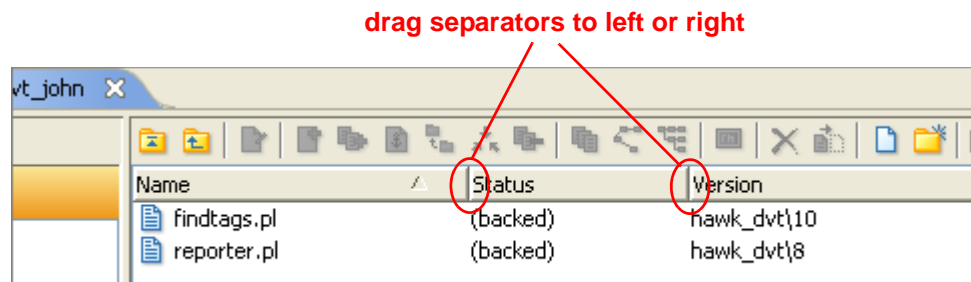
## Working with Tables

The AccuRev GUI often presents information in the form of a table, with multiple rows and multiple columns. You can adjust all such tables, to maximize their usefulness, as described in the following sections.

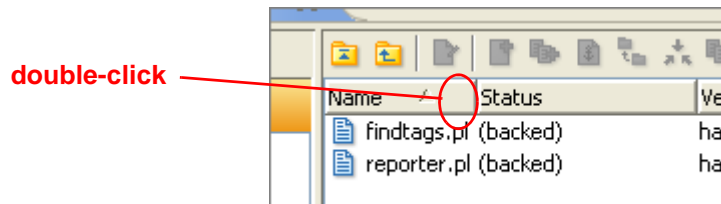
### Adjusting the Widths and Order of Columns

In any table, you can adjust columns widths and change the column order as follows:

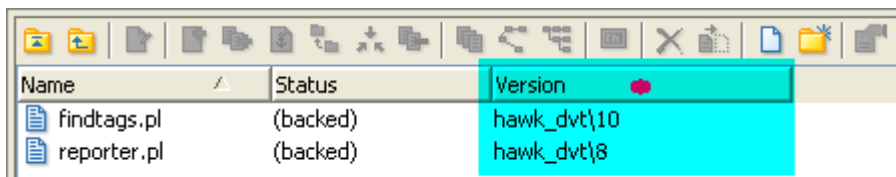
- To resize the columns, click and drag the column separators.



- Double-clicking the column separator to a column's right resizes the column to exactly fit its longest value.

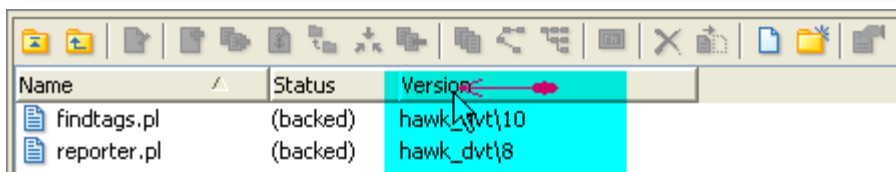


- To rearrange the columns, click and drag the column headers.



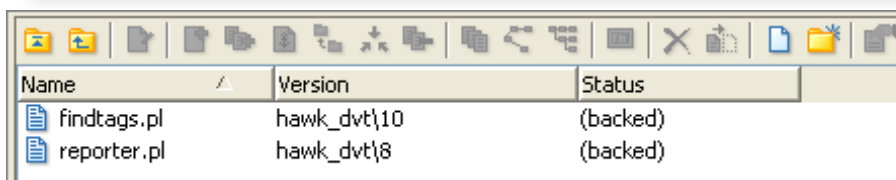
Name	Status	Version
findtags.pl	(backed)	hawk_dvt\10
reporter.pl	(backed)	hawk_dvt\8

1. click column header



Name	Status	Version
findtags.pl	(backed)	hawk_dvt\10
reporter.pl	(backed)	hawk_dvt\8

2. drag column header



Name	Version	Status
findtags.pl	hawk_dvt\10	(backed)
reporter.pl	hawk_dvt\8	(backed)

3. column snaps into place

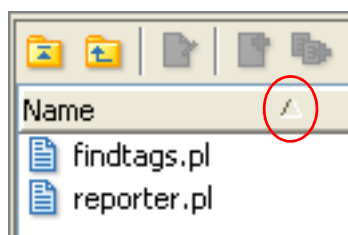
## Sorting the Rows of a Table

Initially, the rows of a table are sorted on one column. A direction icon in the header for that column indicates whether the sort is lowest-to-highest or highest-to-lowest.

This is termed single column mode. Left-click on any column header to continue in this mode:

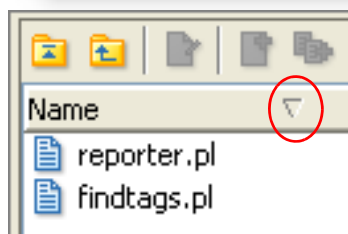
- Click on the current sort column to reverse the sort order.
- Click on another column to make it the sort column.

Note: in the details pane of the File Browser (and any other table that lists file names and directory names in the same column), a lowest-to-highest sort places all directories before all files; a highest-to-lowest sort places all files before all directories.



Name	Status	Version
findtags.pl		
reporter.pl		

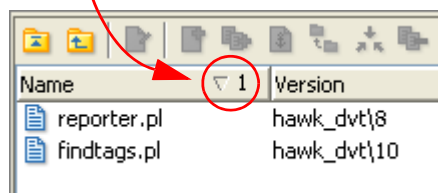
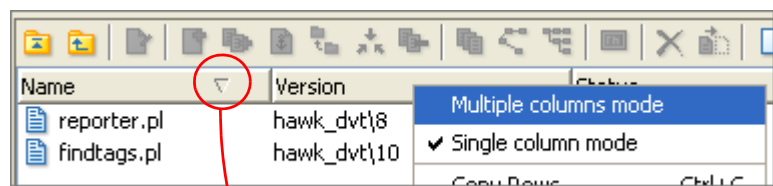
lowest-to-highest



Name	Status	Version
reporter.pl		
findtags.pl		

highest-to-lowest

You can switch at any time to multiple columns mode, in which you define a primary sort column, a secondary sort column, and so on. Right-click any column header to switch sort modes. A “1” appears next to the direction icon in the current sort column, indicating that this is now the primary sort column.

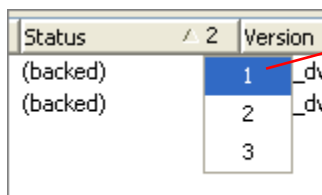


Click another column to make it the secondary sort column. A direction icon annotated with “2” appears in this column, and the rows are reordered according to the two-level sort.

In multiple columns mode, you can:

- Left-click on any column header without a direction icon: this defines an additional sort level, using that column.
- Left-click on any column header that already has a direction icon: this reverses the sort order at that sort level.
- Left-click on any sort column’s level number (1, 2, etc.) to change the sort level of that column.

The rows are reordered automatically each time you perform any of these actions.



left-click  
and select  
new sort  
level

## Using the Keyboard to Navigate through a Table

The keyboard’s navigation keys — up-arrow, down-arrow, PgUp, PgDn, Ctrl-PgUp, Ctrl-PgDn — move a table’s selection highlight in the expected way. In addition, you can navigate by typing any alphanumeric character; this moves the selection highlight to the next row whose entry in the (primary) sort column begins with that character.

## User Preferences

The **Tools > Preferences** command opens a tabbed dialog box, in which you can manage a set of parameters that influence the way the AccuRev looks and works. The preferences you set here apply only to you. They are stored in file **preferences.xml**, in subdirectory **.accurev** of your home directory.

This file is also updated by the **Tools > Change Active User** command.


## ‘General’ Page

### Enable AccuRev Look&Feel

Controls the appearance of GUI tabs. The AccuRev Look&Feel adds an Eclipse-style “X” icon to each tab control, making it easy to close the tab. If a tab is not currently visible (on top of the tab stack), the “X” control appears when the mouse pointer passes over the tab control.

### Enable Issue Preview

Controls the automatic inclusion of an Edit Form pane below the Query Results pane in a AccuWork Queries tab. The contents of the issue record that is currently selected in the Query Results pane is automatically displayed in the Edit Form pane.

The toolbar of the Query Results pane includes a  **Show Issue Form** button, which toggles the inclusion of the Edit Form pane.

Despite the name “Preview”, the Edit Form pane is fully-functional: you can revise and export the contents of issue records in this pane.

### Show Update Log

Controls whether the Update dialog box includes a scrollable text field that displays the names of up to 2000 elements being updated. A complete log of the command’s work is always available, through the **View Full Log** button in the dialog box.

### Show External Elements

Controls whether the File Browser’s details pane includes files and directories that have not been placed under version control (**Add to Depot** command). This setting applies both to the contents of directories (when you’re working in the Folders pane) and to the results of full-workspace searches (when you’re working in the Searches pane).

### Confirm On Exit

Controls whether the AccuRev GUI prompts for confirmation when you invoke the **File > Exit** command or attempt to shut the GUI window.

### Require Manual Refresh

If this preference is cleared, the data displayed on a “buried” GUI tab is automatically refreshed, if necessary, when you switch to that tab. If this preference is checked, the data is not refreshed automatically — you can use the **View > Refresh** command (or function key **F5**) to do so.

### Display of element name in tables

Controls how element pathnames are displayed in tables: in a single **Element** column, or in separate **Name** and **In Folder** columns.

## ‘Diff/Merge’ Page

### Diff

Specifies the graphical file-comparison tool to be invoked by the GUI’s **Diff** command. You can choose AccuRev’s own tool or one of the supported third-party tools from the combo-box. Alternatively, you can type a command line to be executed when the **Diff** command is invoked. See *Enabling the Diff and Merge Tools* on page 115.

### Merge

Similar to the Diff setting — specifies the graphical text-file merge tool by be invoked by the GUI’s **Merge** command.

### Tab size

(AccuRev’s graphical Diff and Merge tools only) The number of spaces to be displayed for each TAB character.

### Ignore Whitespace

(AccuRev’s graphical Diff and Merge tools only) Controls whether whitespace is taken into account when comparing text lines.

### Ignore Changes in Whitespace

(AccuRev’s graphical Diff and Merge tools only) Controls whether a change in the amount of whitespace in a text line is considered to be a change to that line.

### Ignore Case

(AccuRev’s graphical Diff and Merge tools only) Controls whether uppercase and lowercase characters are considered to be the same when comparing text lines.

## ‘Version Browser’ Page

### Initial display mode


Specifies the mode that a new Version Browser tab begins in. **Basic** mode displays “important” versions only, for example omitting intermediate versions in workspace streams. **Expanded** mode displays all versions. Buttons at the bottom of the Version Browser tab switch between the two display modes.

### Initial transaction count

How many of the most recent transactions (and equivalently, versions) a new Version Browser should display. Controls in the Version Browser toolbar change this count.

## ‘Stream Browser’ Page

### Display Default Group in Stream Browser

Controls whether the StreamBrowser includes a  default group control for each stream and workspace with active elements (non-empty default groups). Disabling this feature can significantly improve StreamBrowser performance.

### Enable Stream Browser History

Controls the inclusion in the StreamBrowser toolbar of controls that enable you to “turn back the clock”, viewing a depot’s stream hierarchy as it existed at any point in the past. You must refresh any existing StreamBrowser tabs to make a new setting effective.

## Executing AccuRev Commands

Many AccuRev commands are invoked in the standard “noun-verb” manner: you select one or more files (or other objects); then, you select the command to be executed on them.

### Selecting Objects

The object-selection gestures are the same ones used by Windows Explorer:

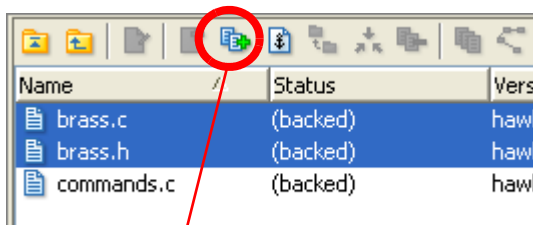
- To select an object, click it with the left mouse button.
- To select a contiguous range of objects listed in a table, click-and-drag with the left mouse button.
- To add or delete an object from an existing selection, hold down the **Ctrl** key and click that object with the left mouse button.
- To extend an existing selection to a particular object, hold down the **Shift** key and click that object with the left mouse button.
- To open a file with a text editor, double-click it with the left mouse button.

### Selecting a Command

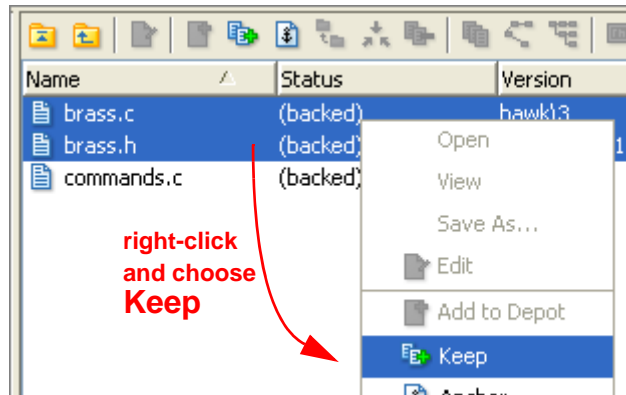
After you’ve selected one or more objects, you execute a command on those objects in any of these ways:

- By clicking a button in the toolbar of the particular GUI tool you’re using.
- By right-clicking the object(s) to display a “context menu” of commands, then selecting one of the commands.
- By opening the **Actions** submenu of the GUI’s main menu, then selecting one of the commands. At any given time, this submenu contains the same commands as the context menu of the currently selected object(s).

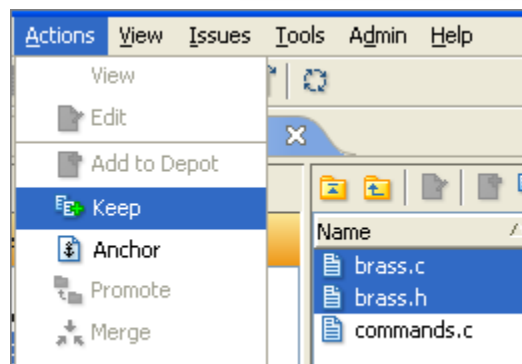




after selecting files,  
click **Keep** toolbar button



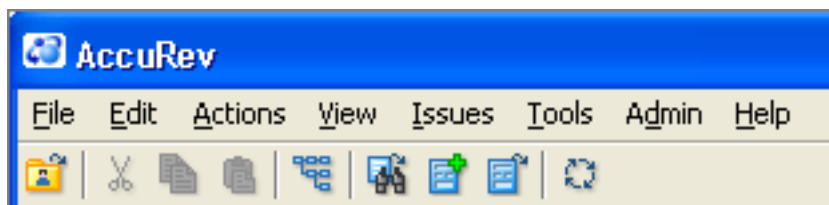
right-click  
and choose  
**Keep**



after selecting files,  
choose **Keep** from  
**Actions** menu

## Main Menu and Toolbar

The GUI window's main menu and toolbar are always visible at the top of the window:



## Main Menu

The following sections describe the commands available through the main menu.

### File Menu

#### New Depot

Create a new depot in the AccuRev repository. You can control the location of the depot's storage directory (also called the slice), whether pathname lookup for the depot's objects will be case-sensitive, and whether the depot's workspaces will use the exclusive file locking

feature. See the *mkdepot* description in the *AccuRev User's Guide (CLI Edition)* for more on these options.

### **New Workspace**

Create a new workspace, backed by a particular stream or snapshot. For details, see *Creating a Workspace* on page 29.

### **New Snapshot**

(enabled when a dynamic stream is selected) Create a new snapshot of a particular stream.

### **New File**

(enabled when you're working in the folders pane of the File Browser) Create a new empty file in the current folder.

### **New Folder**

(enabled when you're working in the folders pane of the File Browser) Create a new empty folder under the current folder.

### **Open**

(enabled when a file is selected in the folders pane of the File Browser) Execute the selected file (like double-click)

### **Open Workspace**

Open a File Browser tab on an existing workspace.

### **Save As**

(enabled when a file is selected in the folders pane of the File Browser) Make a copy of the selected file.

### **Update**

(enabled when the current tab is a File Browser, open on a workspace or reference tree)  
Update the current workspace or reference tree.

### **Update Preview**

(enabled when the current tab is a File Browser, open on a workspace or reference tree) Report which files would be changed by an Update of the current workspace or reference tree.

## Properties

(enabled when an element is selected) Displays a message box containing information about the selected element.

## Close

Close the currently visible GUI tab.

## Close All

Close all GUI tabs.

## Clone

Create another GUI tab that shows that same data structure as the currently visible tab.

## Exit

Quit the AccuRev GUI program.

## Edit Menu

### Cut

(enabled in Merge tool and AccuWork edit forms) Cut the selected text to the clipboard.

### Copy

(enabled in Merge tool and AccuWork edit forms) Copy the selected text to the clipboard.

### Paste

(enabled in Merge tool and AccuWork edit forms) Paste cut/copied text on the clipboard. at the current cursor location.

### Select All

Expand the current selection to include all objects.

### Rename

(enabled when an element is selected in the details pane of the File Browser) Rename the currently selected element.

### Delete

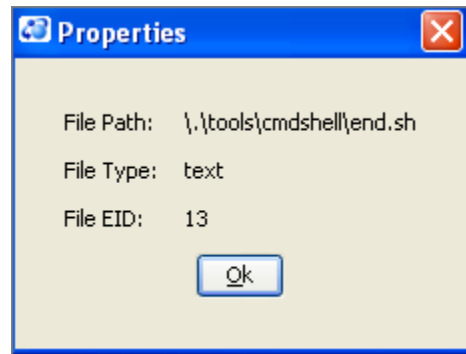
Delete the currently selected file(s) from the workspace tree (disk storage on the client machine). This does not affect the workspace stream (in the repository on the server machine) or anything else in the depot.

### Search

(enabled in AccuRev Diff and Merge tools) Search for a text string.

### Search Again

(enabled in AccuRev Diff and Merge tools) Repeat the most recent text-string search.



## Actions Menu

At any given time, the **Actions** submenu contains the same commands as the context menu of the currently selected object(s).

## View Menu

### Workspaces

List all your workspaces, or everyone's workspaces. From this list, you can open a workspace, hide ("remove") a workspace, or modify a workspace's specifications.

### Streams

Open a StreamBrowser tab, showing all streams in a particular depot.

### Refresh

Bring the data displayed on the current GUI tab up to date.

## Issues Menu

### Queries

Open a Queries tab, in which you create and execute queries on the issue records stored in a particular depot.

### New Issue

Open an edit form to create a new AccuWork issue record in the current depot.

### Look Up

Prompt for an issue number, then open an edit form containing the specified existing AccuWork issue record in the current depot.

## Tools Menu

### Synchronize Time

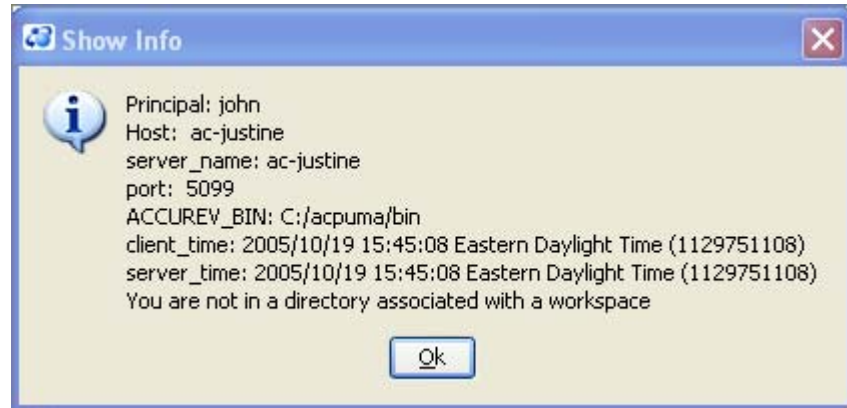
Change the system clock on the client machine to match the clock on the AccuRev server machine. (Succeeds only if the operating system grants you the right to make this change.)

## Info

Display information on the current AccuRev context.

## Server Tasks

Display a table showing the subtasks currently running in the multitasking AccuRev Server.



## Change Active User

Switch to another principal-name. The status indicator in the GUI window's lower left corner is updated. The new principal-name is recorded as the AC\_PRINCIPAL setting in the **preferences.xml** file. See [User Preferences](#) on page 6.

## Change Active Server

Connect the GUI client to another AccuRev Server process and its repository.

## Preferences

Open the AccuRev Preferences dialog box. See [User Preferences](#) on page 6.

## Admin Menu

### Depots

List all the depots in the AccuRev repository. From this list, you can open a depot, display a depot's transaction history, or rename a depot.

### Reference Trees

List all the reference trees in the AccuRev repository. From this list, you can open a reference tree, deactivate ("remove") a reference tree, or modify a reference tree's specifications.

### Triggers

List all the triggers defined for a particular depot.

### Slices

List all the slices (depot storage directories) in the AccuRev repository. Cross-reference this information with a **Depots** listing.

### Locks

List all the stream locks, across all depots in the AccuRev repository. From this list, you can revise or clear individual locks. (To create a new stream lock, use **View > Streams** to open a StreamBrowser for the desired depot.)

## Security

Open a Security tab, whose subtabs provide tools for maintaining users and their passwords, user groups, and access control lists (ACLs) for streams and depots. For more information, see the descriptions of the corresponding CLI commands (**mkuser**, **chuser**, **remove user**, **mkgroup**, **addmember**, **remove group**, **setacl**, **lsacl**) in the *AccuRev User's Guide (CLI Edition)*.

## Schema Editor

Open a AccuWork Schema Editor tab for the issues database in a specified depot. See *Designing Issues Databases and Edit Forms: The Schema Editor* on page 189 of the *AccuWork Issue Management Manual*.

## Help Menu

### Documentation

Display the various books in the AccuRev documentation set (PDF format).

### Quick Setup

Launch a wizard that creates and populates a new depot.

### About

Display program version information on the AccuRev GUI.

## Main Toolbar

The toolbar below the main menu is always available. Note, however, that some of the commands are not valid in all contexts.

### Open Workspace

Open a File Browser tab, showing the contents of a selected workspace.

### Cut

(enabled in Merge tool and AccuWork edit forms) Cut the selected text to the clipboard.

### Copy

(enabled in Merge tool and AccuWork edit forms) Copy the selected text to the clipboard.

### Paste

(enabled in Merge tool and AccuWork edit forms) Paste cut/copied text on the clipboard. at the current cursor location.

### View Streams

Open a StreamBrowser tab, displaying the hierarchy of streams, snapshots, and workspaces in a selected depot.

## Queries

Open a Queries tab, on which you create and execute queries on the AccuWork issue records stored in a selected depot.

## New Issue

Open an empty edit form, on which you create a new AccuWork issue record in the current depot.

## Look Up

Open the AccuWork issue record with the specified issue number in the current depot.

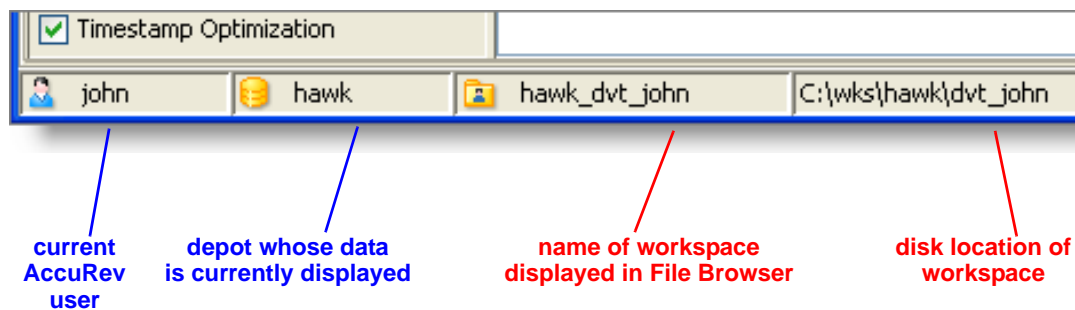
## Refresh

Bring the data in the currently displayed GUI tab up to date.

## Overall Status Indicators

A set of indicators at the bottom of the GUI window show the current context:

*(items in red are displayed if the current tab is a File Browser)*



## The GUI Tools

Subsequent chapters in this manual describe each of the AccuRev GUI's tools:

### File Browser

Displays the files/versions in a workspace or stream; performs searches across the entire workspace or stream, based on elements' AccuRev status.

### StreamBrowser

Displays the hierarchy of streams, snapshots, and workspaces in the current depot, either graphically or in a table (or both). You can drag-and-drop to change the hierarchy or propagate versions up the hierarchy. You can also perform a variety of other operations on the streams, snapshots, and workspaces, including comparisons based on version-IDs or on change packages.

**Diff / Merge tool**

A graphical tool for comparing or merging two versions of a text-file element.

**History Browser**

Displays the transactions related to a particular element, stream, snapshot, workspace, or entire depot.

**Version Browser**

Displays a directed graph, showing some or all the versions of a particular element.

**Stream Version Browser**

Displays a depot's stream hierarchy, either graphically or in a table (or both). In this display, each stream represents the version of a particular element that is currently in that stream.

**Change Palette**

A tool for cross-promoting versions: promoting to a stream other than the parent stream.





## Day-to-Day Usage of AccuRev

This document presents enough information for the individual user to work with AccuRev on a day-to-day basis. We provide a brief overview and discuss a handful of commands. It's a short chapter, because AccuRev is an elegantly simple configuration management system.

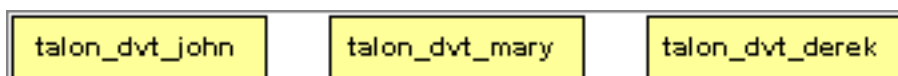
For a more complete overview of AccuRev usage, see *Sample Development Workflow* on page 29.

### The AccuRev Usage Model

AccuRev's flexibility makes it easy to use for a variety of development scenarios. But like every software system, AccuRev has usage models that were foremost in the minds of its architects. This section describes the most common usage model.

AccuRev is a configuration management (CM) system, designed for use by a team of people (users) who are developing a set of files. This set of files might contain source code in any programming language, images, technical and marketing documents, audio/video tracks, etc. The files — and the directories in which the files reside — are said to be “version-controlled” or “under source control”.

For maximum productivity, the team's users must be able to work independently of each other — sometimes for just a few hours or days, other times for many weeks. Accordingly, each user has his own private copy of all the version-controlled files. The private copies are stored on the user's own machine (or perhaps in the user's private area on a public machine), in a directory tree called a workspace. We can picture the independent workspaces for a three-user team as follows:



This set of users' workspaces uses the convention of having like names, suffixed with the individual usernames. AccuRev enforces this username-suffix convention. **talon\_dvt** might mean “development work on the Talon product”; **john**, **mary**, and **derek** would be the users' operating system login names.

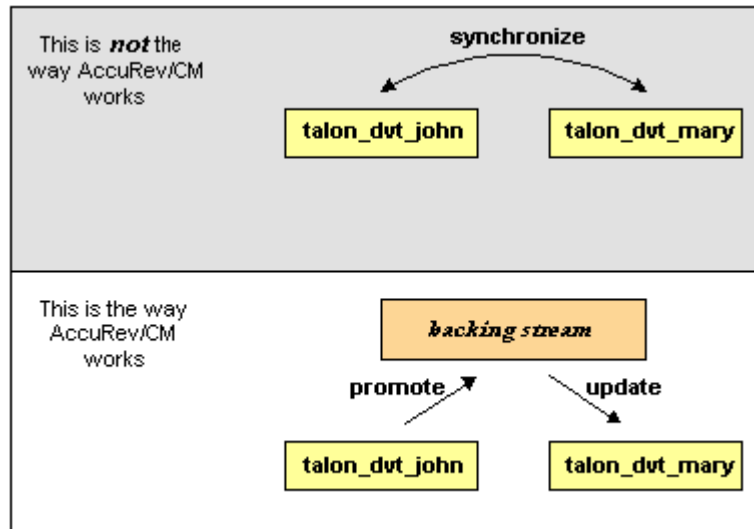
From AccuRev's perspective, development work in this set of workspaces is a continual back-and-forth between “getting in sync” and “getting out of sync”:

- Initially, the workspaces are completely synchronized: they all have copies of the same set of version-controlled files.
- The workspaces lose synchronization as each user makes changes to some of the files.
- Periodically, users share their changes with each other. When **john** incorporates some or all of **mary**'s changes into his workspace, their two workspaces become more closely (perhaps completely) synchronized.

You might assume that the workspace synchronization process involves the direct transfer of data from one workspace to another. But this is not the way AccuRev organizes the work environment. Instead of transferring data directly between private areas (that is, between users' workspaces), AccuRev organizes the data transfer into two steps:

1. One user makes his changes public — available to all the other members of his team. This step is called promotion.
2. Whenever they wish, other team members incorporate the public changes into their own workspaces. This step is called updating.

The first step involves a public data area, called a stream. AccuRev has several kinds of streams; the kind that we're discussing here is called a backing stream. We'll see below how the data in this public stream “is in back of” or “provides a backstop for” all the private workspaces of the team members.




## Change and Synchronization: The Four Basic Commands

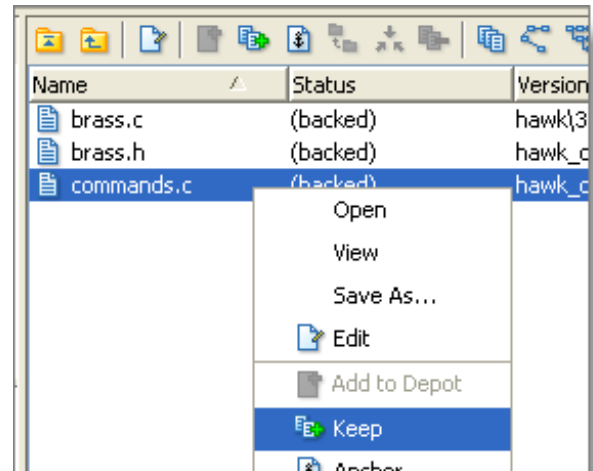
With the usage model described above, you'll be able to accomplish most of your AccuRev work with four simple commands: **Keep**, **Promote**, **Update**, and **Merge**. We describe these commands in the following sections. Each section has a subsection titled “The Fine Print”, in which we present additional usage details, notes on the way AccuRev implements certain features, and other tidbits of interest. You might want to skip over these sections on your first reading of this material.

### Keep: Preserving Changes in Your Private Workspace

An AccuRev workspace is just a normal directory tree, in which you make changes to version-controlled files. You can work with the files using text editors, build and test tools, IDEs, etc., just as if the files weren't version-controlled at all. For example, you might edit a source file and invoke the editor's “Save” command a dozen times over the course of an hour or two. These operations don't involve AccuRev at all — they simply have the operating system change the contents and the timestamp of the file in your workspace.

You don't need to perform a “check out” operation or otherwise get permission from AccuRev before editing a file in your workspace. (Some legacy CM systems do impose such a regimen.)

Every so often, you want AccuRev to preserve the current contents of the file as an official new version of the file. You accomplish this using AccuRev's **Keep** command. This figure shows how to invoke the **Keep** command from a file's context (right-click) menu in the AccuRev File Browser tool, which has a Windows Explorer-like interface. You can also invoke **Keep** with the  toolbar button.



You can continue modifying the file, then using **Keep** to preserve the latest changes, as often as you like. Other team members won't complain about "thrashing", because these new versions stay within your workspace; without affecting any other user's workspace.

AccuRev retains all the versions that you **Keep**. This makes it possible for you to roll back to any previous version you created.

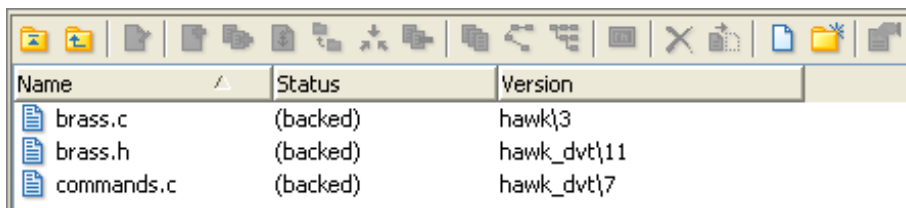
Several other operations are similar to **Keep**, in that they create a new version of a file in your workspace, without affecting any other user's workspace. The most important are:

- **Rename/Move**: You can rename a file or move it to a different directory (or both), using AccuRev commands. Other users will continue to see the file at its original pathname in their workspaces.
- **Defunct**: You can remove a file from your workspace with the AccuRev command **Defunct**. Other users will continue to see the file in their workspaces.

## The Fine Print

We said above that AccuRev "retains all the versions that you **Keep**". But where? Each time you **Keep** a file, its current contents are copied to the AccuRev repository, located on the machine where the AccuRev Server runs. You don't need to care about the name and precise location of this copy. Each version you create has a version-ID, such as **talon\_dvt\_john/12** ("the 12th version of this file created in workspace **talon\_dvt\_john**").


AccuRev keeps track of the status of each file in a workspace. After you **Keep** a file, the Status column in the AccuRev File Browser contains the indicator (**kept**). It also contains the indicator (**member**), meaning that the file belongs to the set of files you're actively working on. (See **Active and Inactive Files** below.) The Version column displays the version-ID.

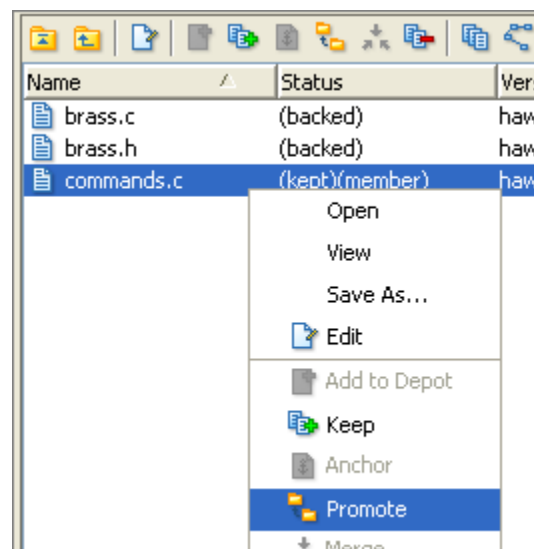


A change to the data within a file, recorded by **Keep**, is termed a content change; the change made by **Rename/Move** or **Defunct** is termed a namespace change. (Many CM systems don't handle namespace changes at all, or have very limited capabilities in this area.) As noted above, AccuRev saves a new copy of the file in the repository whenever you make a content change. But it doesn't need to copy the file when you make a namespace change; rather, the AccuRev Server just records the change in its database.

To perform version control on directories, AccuRev only needs to keep track of namespace changes — renaming, moving, or deleting a directory. Unlike some legacy CM systems, AccuRev doesn't need to record a new directory version when you make a content change — for example, adding a new file to the directory.

## **Promote: Making Your Private Changes Public**

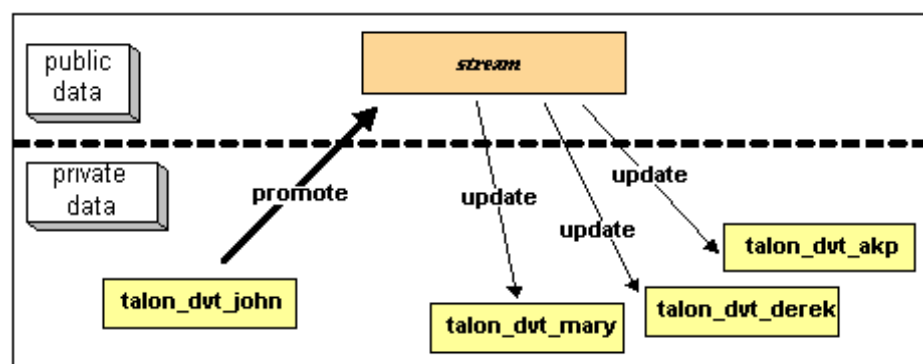
At some point, after you've used **Keep** to create one or more new, private versions of a file in your workspace, you typically want to share the changes you've made with the other team members. To make your (most recent) new version “available to the public”, you promote it. This figure shows how to invoke the **Promote** command from a file's context (right-click) menu in the File Browser. You can also invoke **Promote** with the  toolbar button.



Promoting your new version of a file does not automatically “push” it into the workspaces of the other team members. When a user decides that he's ready to incorporate versions of files that other team members have **Promoted**, he “pulls” them into his workspace with the **Update** command (details below).

## **Streams**

The **Promote** command sends data to — and the **Update** command gets data from — a sophisticated AccuRev data structure called a stream. The stream acts as a “central data exchange” for the set of workspaces used by a development team. A stream also has a bit of “traffic cop” built in,



preventing team members' efforts from colliding and providing other mechanisms to control the flow of data.

A stream is not, as you might initially suppose, a set of copies of promoted files. Rather, it's more like a list of version-IDs.

- the 4th version created in workspace **talon\_dvt\_john** of file **command.c**
- the 7th version created in workspace **talon\_dvt\_mary** of file **brass.c**
- ... etc.

In CM-speak, a stream is a configuration of a collection of version-controlled files. The term “stream” is apt, because it implies the ongoing change of a development project. Each time a user promotes a version of file **brass.c**, the stream configuration changes for that file — for example, from “the 5th version created in workspace **talon\_dvt\_derek**” to “the 7th version created in workspace **talon\_dvt\_mary**”.

### Promotion and Parallel Development

Sometimes, AccuRev doesn't allow you to promote a file to the development team's stream, because another team member has already promoted the same file (after modifying it and performing a **Keep** on it). AccuRev is preventing you from overwriting your colleague's change to the team's shared stream. This situation is called an overlap: two users working at the same time on the same goal, to create the stream's next version of a particular file.

Before you can promote your changes to the stream, you must first perform a merge on the file that has an overlap (details below).

### Active and Inactive Files

As you work with a file using the commands described above, AccuRev considers the file to alternate between being *active* in your workspace and *inactive*:

- The file is initially *inactive*.
- When you create a new version in your workspace, using **Keep**, **Rename/Move**, or **Defunct**, the file becomes *active*.
- When you make your private version public, using the **Promote** command, the file becomes *inactive* again.

Later, you might restart this cycle, making the file active again by creating another new version of it. Alternatively, an update of your workspace might overwrite your inactive file with a newer version that another team member promoted.

AccuRev keeps track of the set of active files in your workspace. Officially, this set is called the default group. You might find it easier to think of it as the workspace's “active group” or its “active set”.

## The Fine Print

The **Promote** command doesn't copy the promoted version to the AccuRev repository. It doesn't need to. Promotion just gives an additional name to a version that *already exists* in the repository — having been placed there by a previous **Keep** command (or **Rename/Move** or **Defunct**). For example, promoting “the 7th version created in workspace **talon\_dvt\_mary**” might give that version the additional name “the 3rd version promoted to stream **talon\_dvt**”.


Just to emphasize the previous point: a stream does not reside in the file system, but in the database managed by the AccuRev Server. Promoting a version to a stream does not create a copy of a file; it just creates an additional file-reference in the Server database.

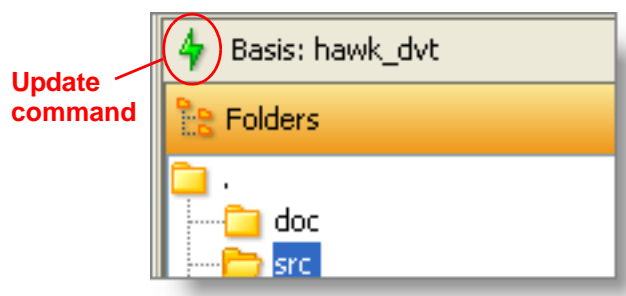
It might seem strange at first that deleting a file with the **Defunct** command makes the file *active*. The File Browser continues to list the file — with a (**defunct**) status — even though the file has been removed from your workspace's disk storage. This design feature enables AccuRev to implement the file-deletion operation using the same private-change/public-change scheme as all other changes.

We've discussed *the* stream behind a set of workspaces. But a typical development project has many streams, organized into a hierarchy. Promoting a version to a higher-level stream from a lower-level stream makes that version “even more public” — for example, available to users outside your local development team.

## Update: Incorporating Others' Changes into Your Workspace

As users work independently of each other, the contents of their workspaces increasingly diverge. Typically, some of the differences between workspaces are inconsistencies. For example, changes that John makes in a report-library routine might cause errors in the report program that Mary's writing. To minimize the time and effort required to resolve inconsistencies during the “integration” phase of a project, it makes sense to have users synchronize their workspaces on a regular basis.

With AccuRev, synchronization does not mean incorporating data into your workspace directly from one or more other workspaces. Instead, synchronization involves copying data into the workspace from the stream to which all team members **Promote** their changes. This operation is performed by the **Update** command. This figure shows the  **Update** toolbar button. You can also invoke this command as **File > Update** from the main menu.



Note: the stream's role as a provider of data — through **Updates** — to a set of workspaces motivates the term backing stream. Think of restocking a store's shelves with merchandise retrieved from “the back room”.

So an update operation on your workspace copies versions of certain files from the backing stream to the workspace, overwriting/replacing the files currently in the workspace. But which

files? **Update** changes a file if (1) there is a newer version in the backing stream, and (2) the file is *not* currently active in your workspace.

**Update** won't overwrite an active file, even if there's a new version of it in the backing stream. No matter how good someone else's code is, you don't want his changes to wipe out the changes that you've been making! This situation is another instance of an overlap, which we described in the **Promote** section above. (You can encounter an overlap both (1) if you're trying to make your private changes public (promotion), or (2) if you're trying to bring already-public changes into your private workspace (updating).) In all such situations, AccuRev resolves the overlap situation with a merge operation (details below).

**Update** handles namespace changes as well as content changes. Thus, if your colleague renamed a file and promoted the change, an update will cause the file to be renamed in your workspace. And if your colleague removed a file (**Defunct** command), an update will cause the file to disappear from your workspace.

### The Fine Print

Here's how AccuRev prevents an update from "clobbering" your changes. The first thing **Update** does is to analyze your workspace, determining whether each version-controlled file is "active" or "inactive". Initially, all the files in a workspace are inactive — each one is a copy of some version in the repository. (For each version-controlled file, AccuRev keeps track of *which* particular version.)

A file is deemed to be active in your workspace if you've created a new version of it, using the **Keep**, **Rename/Move**, or **Defunct** command. (A couple of additional commands "activate" a file; one of them is discussed below.) When **Update** copies versions from the repository into your workspace, it skips over all such active files.

Note: **Update** can tell if you've modified a file but have not yet stored the changes in the repository as a new **Keep** version. It uses file sizes, timestamps, and checksums to determine this. The presence of any such files prevents the update from proceeding. You can use the **Anchor** command to activate such files, enabling **Update** to do its work.

### Merge: When Changes Would Collide ...

The preceding sections, on the **Promote** and **Update** commands, both discuss the situation in which two users concurrently work on the same file. Their changes to the file are said to overlap. Both **Promote** and **Update** decline to process a file with overlap status, because doing so would cause one user's changes to overwrite the other's changes.


For example:

- Team members John and Mary both **Keep** one or more new private versions of **brass.c** in their respective workspaces.
- Mary **Promotes** her latest new version of **brass.h** to the backing stream.
- At this point, AccuRev will decline to do either of the following:
  - **Promote** John's version of **brass.h** to the backing stream.



- Overwrite John’s copy of **brass.h** during an update. (The **Update** command skips over this file, but continues its work on other files.)

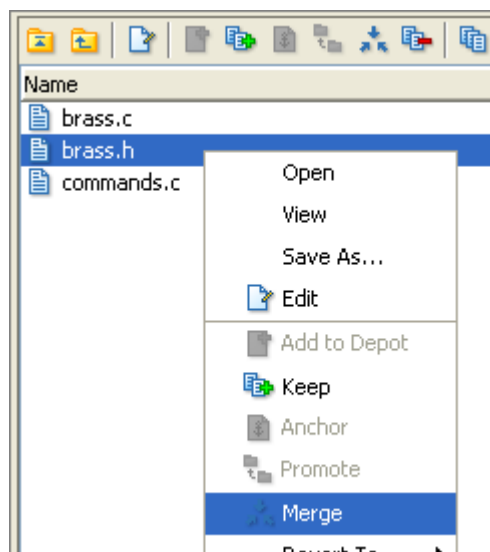
To enable either a promotion or an update of **brass.h**, John must incorporate, or merge, the version in the backing stream — which contains Mary’s changes — into his own copy of the file. The **Merge** command is essentially a fancy text editor, which combines the contents of two versions of a text file. The resulting “merged version” replaces the file in John’s workspace.

This figure shows how to invoke the **Merge** command from a file’s context (right-click) menu in the File Browser. You can also invoke **Merge** with the  toolbar button.

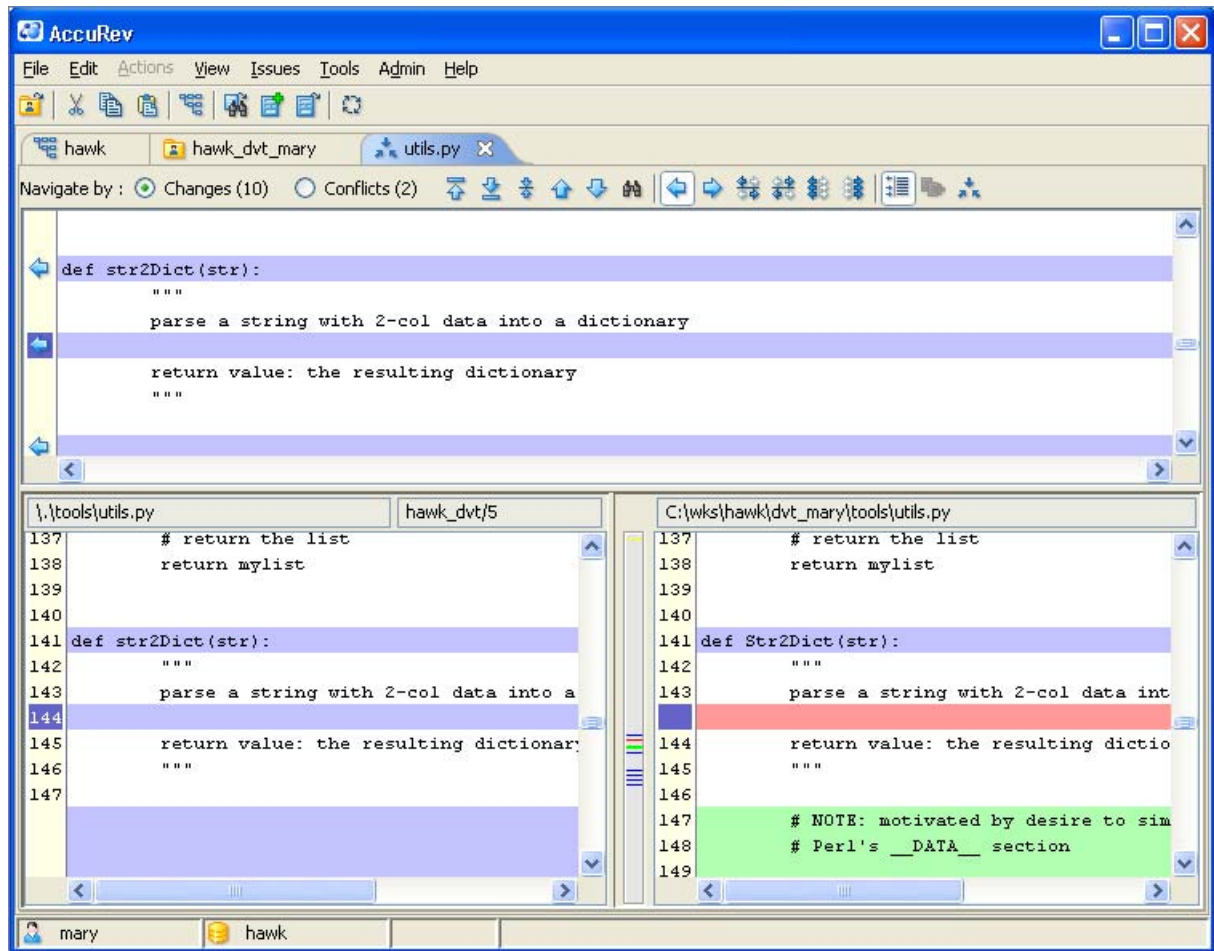
Often, a merge operation is unambiguous, and so can be performed automatically. For example, suppose Mary’s changes to file **brass.h** all occur in lines 1–50, and all of John’s changes occur in lines 125–140. In this case, merging the two versions involves replacing some or all of John’s first 50 lines with Mary’s. Now, the edited version of **brass.h** in John’s workspace contains both users’ changes.

Note: we don’t claim that the two sets of changes are semantically consistent with each other.

That’s what the build-and-test cycle is for!



If both John and Mary have made changes to the same part of the file — say, lines 85–87 — then John must decide how to resolve this conflict. The graphical **Merge** tool makes this easy:



After performing a merge, AccuRev automatically **Keeps** the merged version to preserve the results of the merge operation. You can then **Promote** the merged version to the backing stream. After that, other team members can use **Update** — or perhaps **Merge** — to bring all the changes into their workspaces.

### The Fine Print

The graphical Merge tool performs a “3-way merge”, which uses the common ancestor of the two versions being merged. This algorithm helps to automate the merge operation, often completely eliminating the need for human intervention. AccuRev performs merge operations on text files only, not on binary files.

AccuRev keeps track of all merge operations. This greatly simplifies subsequent merge operations on files that have been merged previously: you don’t need to resolve the same conflicts over and over again.

The most common overlap situation involves AccuRev’s preventing you from promoting a file, because someone else “got there first” in creating a version in the backing stream. AccuRev can also detect deep overlaps, in which another user “got there first” in creating a version in the *parent* of the backing stream, or in other higher-level streams.

## Learning More about AccuRev

Armed with the four commands **Keep**, **Promote**, **Update**, and **Merge**, you'll be able to work effectively in team parallel development environment. To make full use of AccuRev's configuration management capabilities, you'll need to dig a bit deeper. But no matter what your CM challenges are, we think you'll find that AccuRev meets them with an architecture and user interface that are intuitive and easy to learn.

# Sample Development Workflow

This chapter presents a sample workflow for usage of AccuRev's configuration management features. AccuRev does not force you to proceed with your software development work in any particular way. We're here to help you, not put you in a straitjacket! The workflow described in here shows how you can put the AccuRev GUI to good use, but it certainly isn't the only way to get your work done.

We'll assume that a depot has already been set up in the AccuRev repository. Instructions for creating a depot and populating it with files to be version-controlled are included in the "Quick Evaluation" chapters in *AccuRev Technical Notes*.

Our example workflow includes these steps:

- *Creating a Workspace*
- *Navigating a Workspace*
- *Editing Source Files*
- *Checkpointing — Saving Private Versions*
- *Concurrent Development — Incorporating Other Developers' Work*
- *Making Your Changes Public*
- *Concurrent Development — When Streams Collide*
- *Getting in Touch With Your Past*

## Creating a Workspace

A depot contains a hierarchy of streams. Any stream can have one or more workspaces based on ("backed by") it. An AccuRev workspace is a directory hierarchy on your hard disk, containing files that are being managed by AccuRev. You can create any number of workspaces — different ones for different development projects.

Note: usually, you can think of a workspace as simply containing a collection of files.

Sometimes, though, it helps to adopt AccuRev's viewpoint: each file is a version-controlled element; your workspace contains a copy of a certain version of that element.

You can always use the command **File > New > Workspace** on the main menu to create a new workspace. If you're currently working with some existing stream (for example, in the StreamBrowser tool), you can also use the **New Workspace** command on the stream's context menu.

Workspace creation is handled by a wizard: multiple screens that you navigate by clicking **Next** and **Previous** buttons. The following screen shots show the steps:

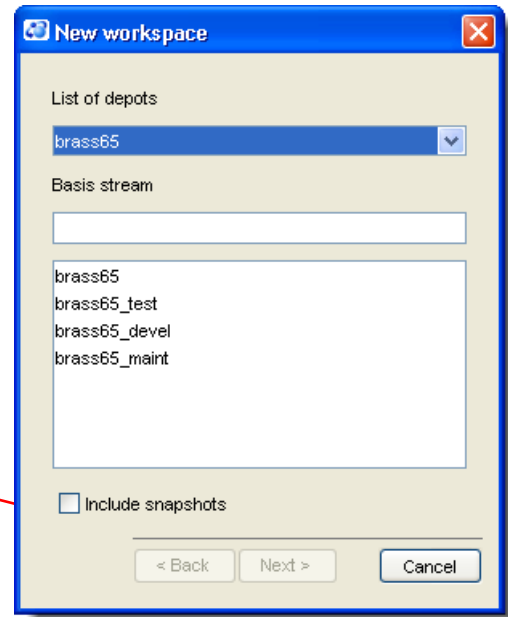
- **Selection of backing stream:**

The first screen depends on whether you have a “current stream” context. If you do, the wizard assumes that you want the current stream to be the backing stream of the new workspace.

Otherwise, the wizard presents a screen on which you select a particular stream in a particular depot, to be the backing stream. You can also choose a snapshot (“frozen stream”) to back a workspace.

list of all  
existing streams in  
selected depot

... and maybe  
snapshots, too



- **Name of Workspace to Create:** Click **Next** to accept the wizard’s offer to name the workspace after the backing stream. The suffix *\_**<principal-name>*** is appended automatically to the workspace name.

This makes it easy to set up a backing stream and a set of like-named workspaces, one for each user:

backing stream:

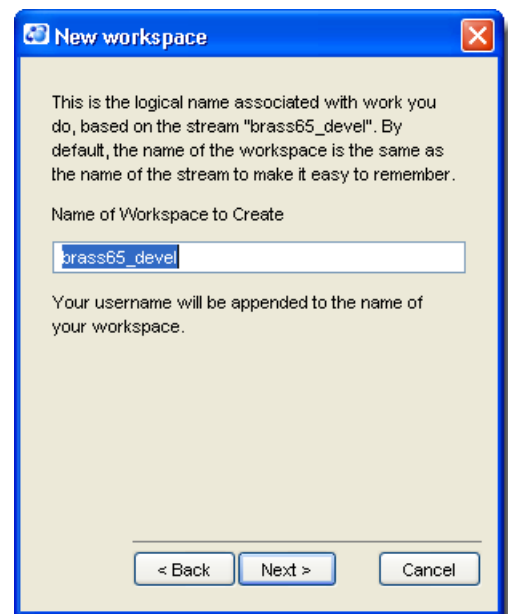
**brass65\_devel**

workspaces:

**brass65\_devel\_mary**

**brass65\_devel\_jjp**

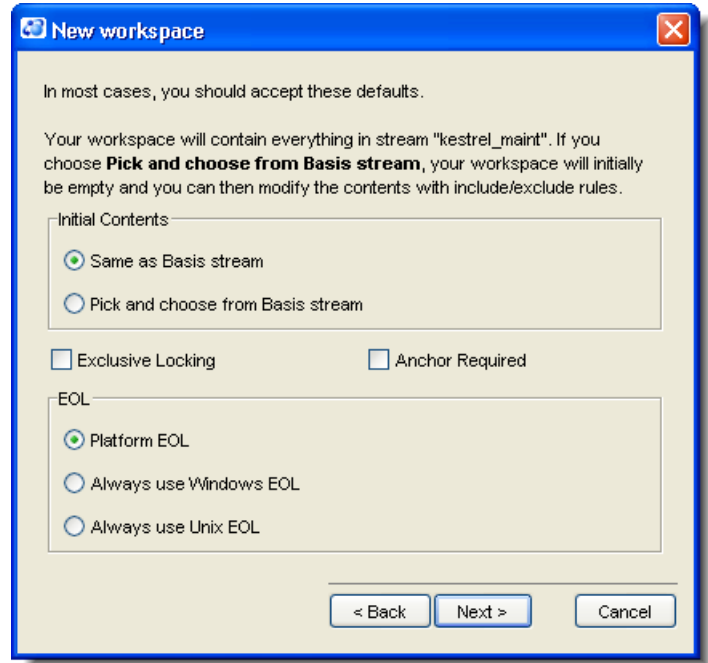
**brass65\_devel\_derek**



- **Workspace options:** Click **Next** to accept the defaults for these workspace options:

- **Initial Contents:** By default, all the elements in the backing (“basis”) stream also appear in the workspace. You can use the Include/Exclude facility to select a particular set of files and directories to appear in the workspace.

- **Type of Workspace:** By default, you can modify any file in a workspace at any time, without having to interact with AccuRev. And several users can work on the same file concurrently in their separate workspaces. But AccuRev also supports more restrictive development processes:



But AccuRev also supports more restrictive development processes:

*Exclusive File Locking:* This feature suppresses the ability of multiple users to work on the same file at the same time. When one user starts working on a file, others are prevented from doing so. By default, this feature is turned off, enabling parallel development in this workspace.

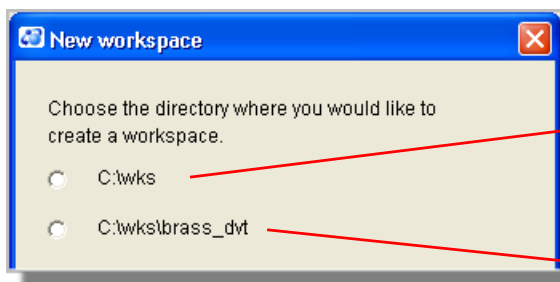
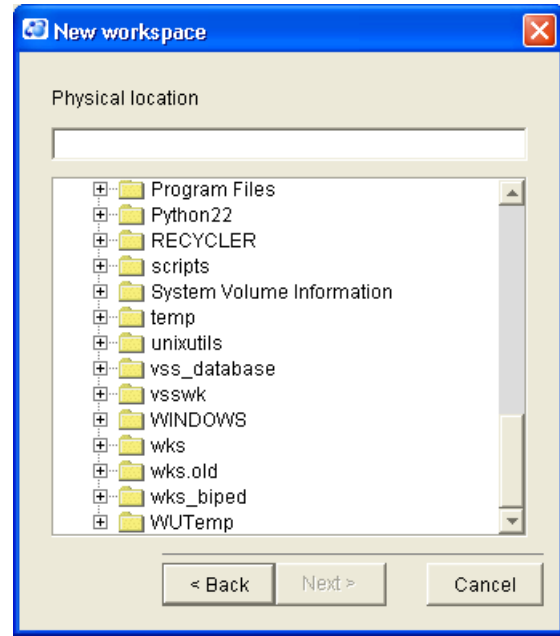
Note: you cannot reparent, rename, or change any of the other settings of a workspace that uses the exclusive file locking feature.

*Anchor Required:* This feature requires you to run an AccuRev command, **Anchor**, before working on a file — even if no one else is currently working on it.

- **Text-file line terminators:** By default, AccuRev uses the line terminator appropriate for the client machine’s operating system. You can force Unix or Windows line terminators.

- **Physical Location:** You can create an empty workspace, or you convert an existing source tree into a workspace. In either case, navigate to an existing directory. Then, click **Next**. The wizard lets you choose whether to:

- Convert the existing directory into a workspace
- Create an empty workspace in a subdirectory of the existing directory. (The wizard automatically chooses a name for the subdirectory.)



convert existing  
source tree  
into workspace

create empty workspace  
in new subdirectory

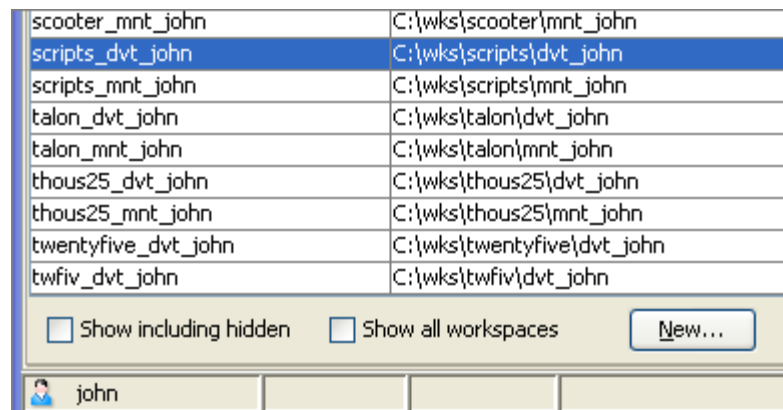
Select your choice, then click **Finish**.

When the New Workspace wizard finishes, it automatically opens a File Browser tab, showing the contents of the workspace.

## Navigating a Workspace

If you've just created a workspace. AccuRev automatically opens a File Browser on it. Otherwise, you can go to an existing workspace as follows:

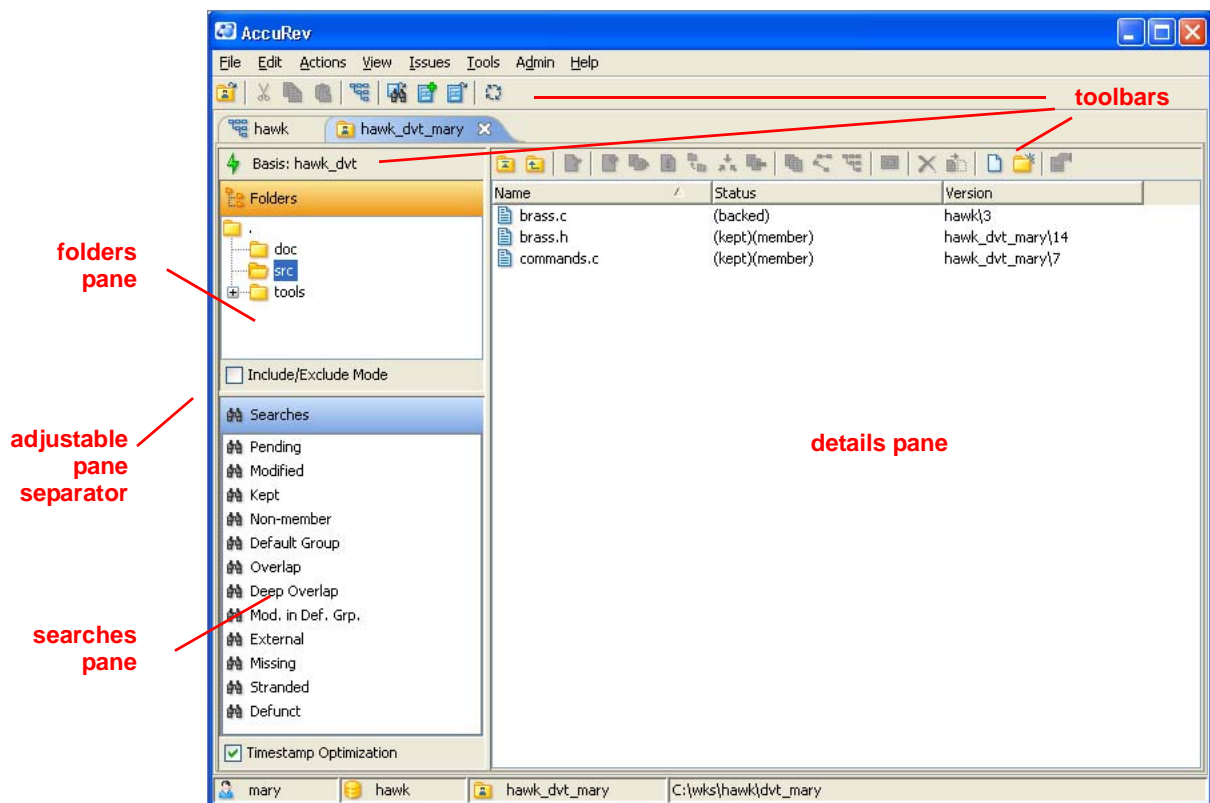
- Select **View > Workspaces** from the command menu.



- In the **Workspaces** tab, right-click the desired workspace, and select **Open** from the context menu.

By default, the **Workspaces** tab shows only workspaces that belong to you. To see workspaces that belong to other users, use the **Show all workspaces** checkbox at the bottom of the tab.

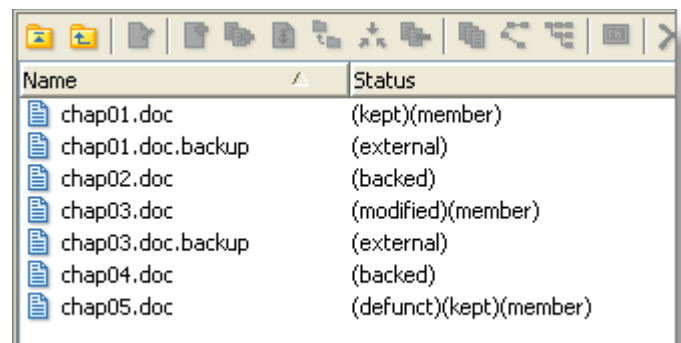
The File Browser tool makes it easy to view, monitor the status, and change the contents of files located in workspaces. Its display resembles that of Windows Explorer. To the familiar folders (navigation) and details panes, adds a unique searches pane, which you can use to search throughout an entire workspace for files in various stages of development. Each of these panes has its own toolbar, separate from the GUI's main toolbar.



By default, the details pane displays the files in one directory (folder) — the one currently selected in the folders pane. For each file, the **Status** column shows one or more status flags.

The **(backed)** flag means that you haven't made any changes to the file. The **(kept)** flag means that you've made some changes, and have stored them in a "private" version, visible only in this workspace. The **(modified)** flag means that there are changes to the file that have not been "kept".

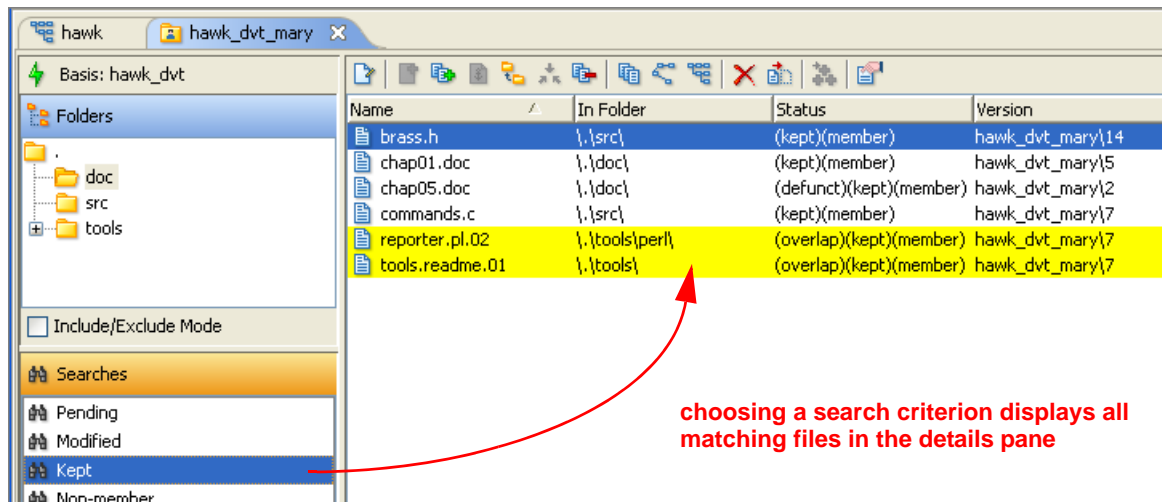
(For more on status flags, see [AccuRev File Statuses](#) on page 48.)





The searches pane enables you to see “just the files you care about”, instead of “all the files”. Which files do you care about? In general, a workspace contains a copy of the entire source base, which might include hundreds, or perhaps thousands, of files. But for a given development project, you’ll probably modify only a handful of the files. You need the others for general reference, and to enable you to perform software builds and tests in the workspace.

For example, you may have changed files in several different directories, and for each one kept a private version. So you may “care about” all the files in the workspace whose status flag is (**kept**). Use the **Kept** search to see exactly the information you want:



searches pane  
contains list of  
search criteria

Note that the **Kept** search displays complete pathnames within the workspace hierarchy. That’s because a searches processes the entire workspace, not just in a particular subdirectory.

Similarly, the **Modified** search makes it simple to locate all files in the workspace that have a (**modified**) status flag. The GUI provides quite a few search criteria; you can always locate all the files you care about, simply and quickly.

## Editing Source Files

To edit a file, right-click it in the details pane and select **Edit** from the context menu. A double-click might bring up a text editor, but it might attempt to execute the file as a program instead. (Either way, be sure to click on the name, not on the icon.)


You can also start up a text editor or integrated development environment (IDE) independently of the AccuRev GUI, and edit files in that context.

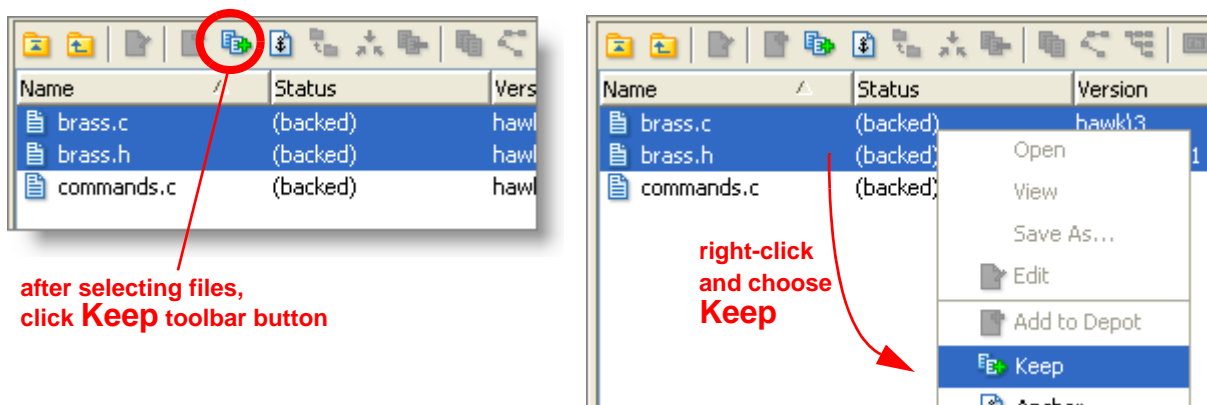
After saving your changes to the text, you may need to invoke the **View > Refresh** command in the GUI window, in order to update the files’ status flags. Function key **F5** is a keyboard shortcut to this command.

## Checkpointing — Saving Private Versions

At any time, you can keep the changes you've made in one or more files. The **Keep** command creates an official new version of a file. This includes making a permanent copy in the depot of the file's current contents. At any point in the future, you can revert to this version. This “save it just in case” procedure is commonly called checkpointing.


But **Keep** does not make the new version public — it remains private to your workspace. Nobody else will see your changes yet. You can keep as many private versions (i.e. checkpoint the file as many times) as you want, without affecting or disrupting other people's work.

You can invoke the **Keep** command from the context menu of a selected file (or group of files), or with the  **Keep** toolbar button:



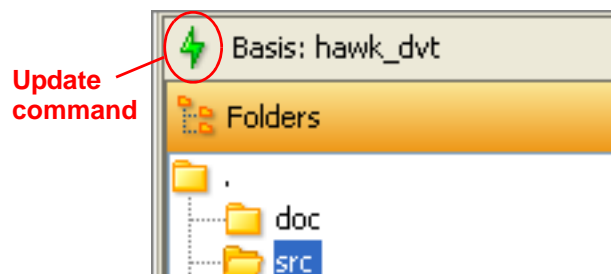
If the files you wish to keep reside in several different directories, you can select all of them at once, using a whole-workspace search for **Modified** files. (See *Working in the Searches Pane* on page 73.)

## Concurrent Development — Incorporating Other Developers' Work


To incorporate other people's changes into your workspace, click the  **Update** button, located above the folders pane.

You won't get the changes they have preserved with **Keep** — those changes are private, as explained above. You'll only get the changes they have made public with the **Promote** command. (There's more on **Promote** just below.)

You can update your workspace as often or as infrequently as you wish.

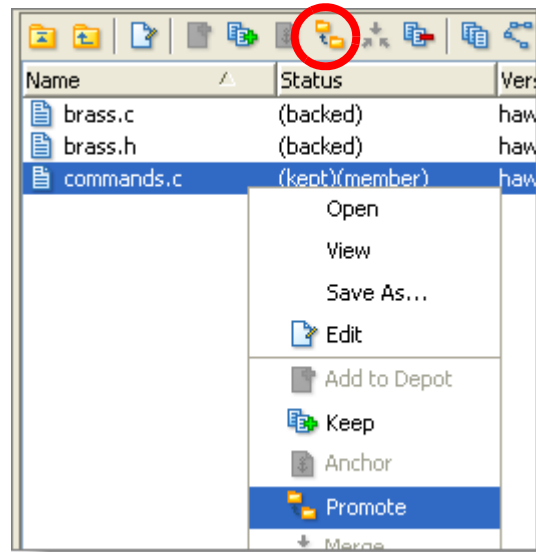


## Making Your Changes Public

To make your work available to others, you promote your kept files. You can invoke the **Promote** command from the context menu of a selected file (or group of files). There is also a  **Promote** toolbar button:

If the files you wish to keep reside in several different directories, you can select all of them at once from the results of a **Kept** or **Pending** search. (See *Working in the Searches Pane* on page 73.)

Promoting a file makes the private version, which you previously created in your workspace with **Keep**, into a public version. The public version resides in your workspace's backing stream. Anyone else whose workspace has the same backing stream can incorporate your promoted versions, using the **Update** command.




## Concurrent Development — When Streams Collide

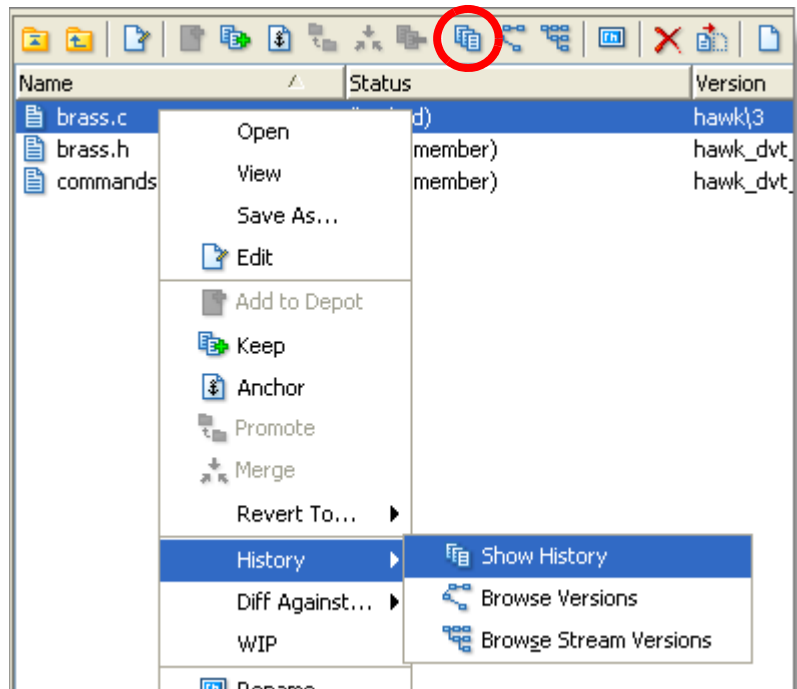
If someone else promotes a change to a file before you do, the changes are said to overlap. Before promoting your changes to the backing stream, you must merge the already-promoted changes into your work. Merging is a big topic; for more on AccuRev's Merge tool, see *The AccuRev Diff, Merge, and Patch Tools* on page 115.

## Getting in Touch With Your Past

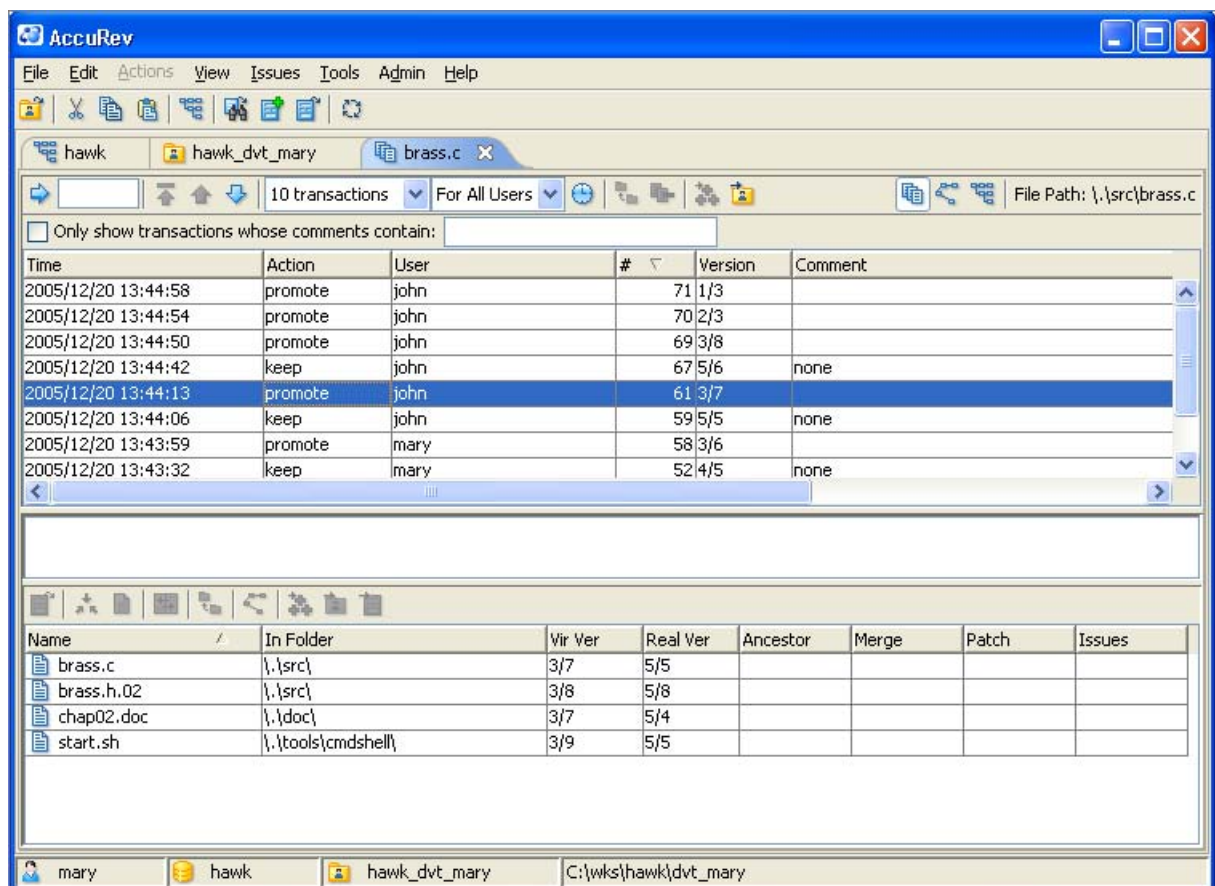
AccuRev keeps track of the complete history of each version-controlled file (or element). This history consists of the set of transactions that involved that particular element. Typically, most of an element's transactions are **Keep** and **Promote** actions. Transactions are also logged in other situations: when an element is first added to the depot (**create**, even though the command-name is **Add to Depot**), when you rename it or move it to a different directory (**move**), when you incorporate someone else's changes into your work (**merge**), etc. In general, any change to a depot gets logged by a transaction.

To view the history of a selected element, invoke the **History > Show History** command from the element's context menu.

Alternatively, use the  **Show History** button in the toolbar of the details pane.




A new tab opens, containing AccuRev's History Browser.

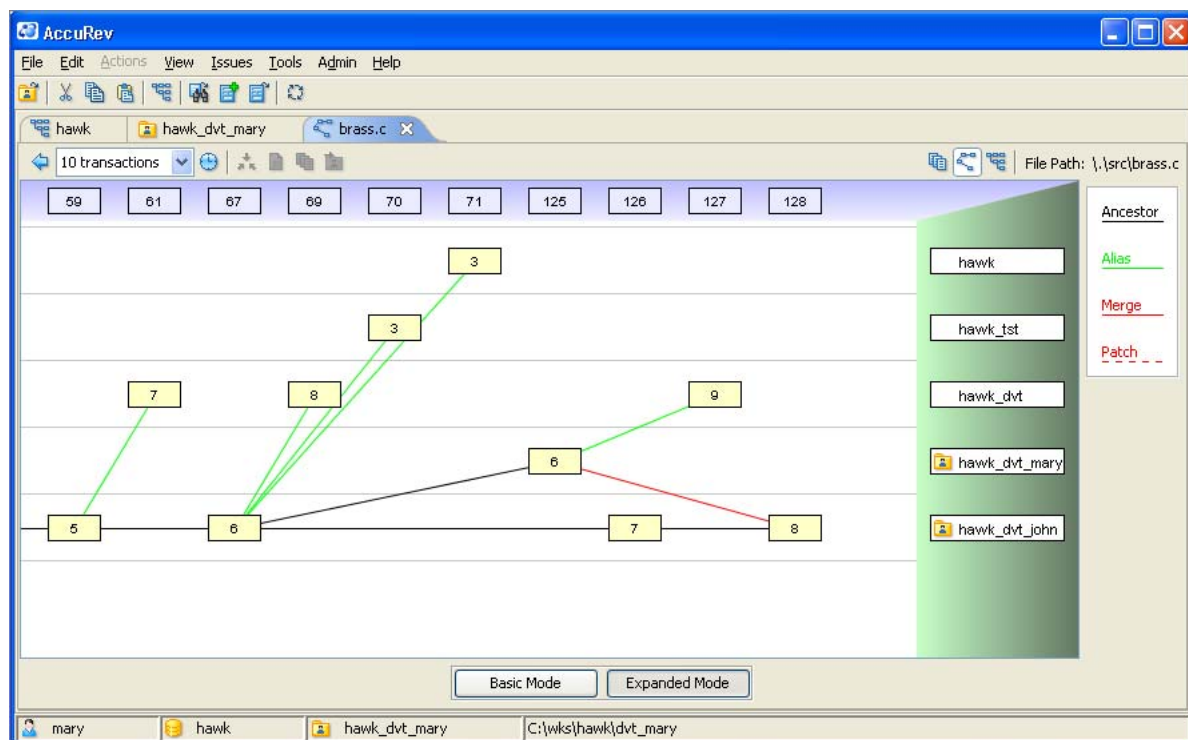


The History Browser tab has three panes. The table in the top pane summarizes the requested transactions, one transaction per row. The middle pane shows the comment string entered by the user who performed the transaction. The bottom pane lists each element involved in the transaction. For more on the History Browser, see *The History Browser* on page 135.

## The Version Browser

You can also view the history of an element graphically. Select **History > Browse Versions** from the element's context menu, or use the  **Browse Versions** toolbar button.

The Version Browser window displays all of an element's versions. This includes private versions you and your colleagues have created in your workspaces (**keep** command), along with public versions that have been placed in the backing stream (**promote** command). In a depot with a sophisticated stream hierarchy, the display may include versions in other streams, as well.



Holding the mouse pointer over a version displays a help balloon containing more information about the version, including the creation log message.

The versions are connected with color-coded lines, indicating the way in which the versions were created:

- black lines indicate direct ancestry (edit existing version, then **Keep** new version)
- green lines indicate promotions between streams
- solid-red lines indicate merges between streams


- dashed-red lines indicate patches — changes made in individual versions, rather than a stream's accumulated changes

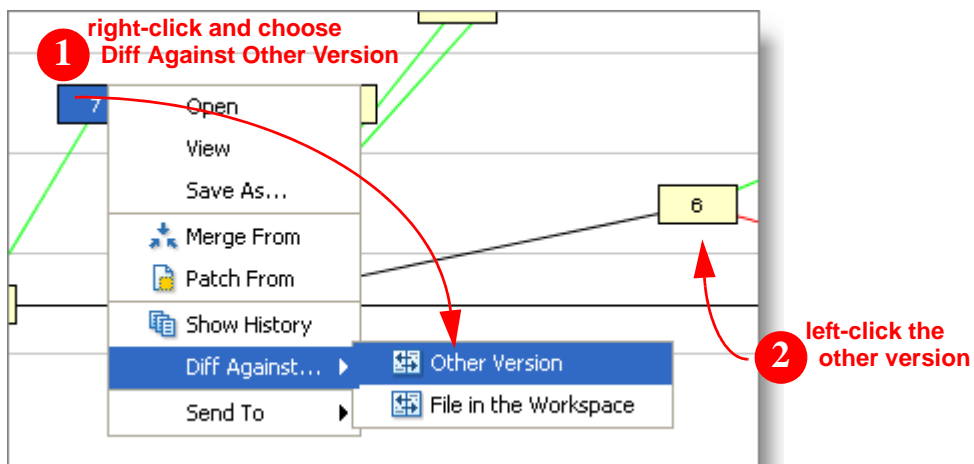
For more on ancestry, see *The Version Browser: Ancestry Tracking* on page 143.

Each version was created in a particular transaction; the white box at the top of the window shows the transaction number.

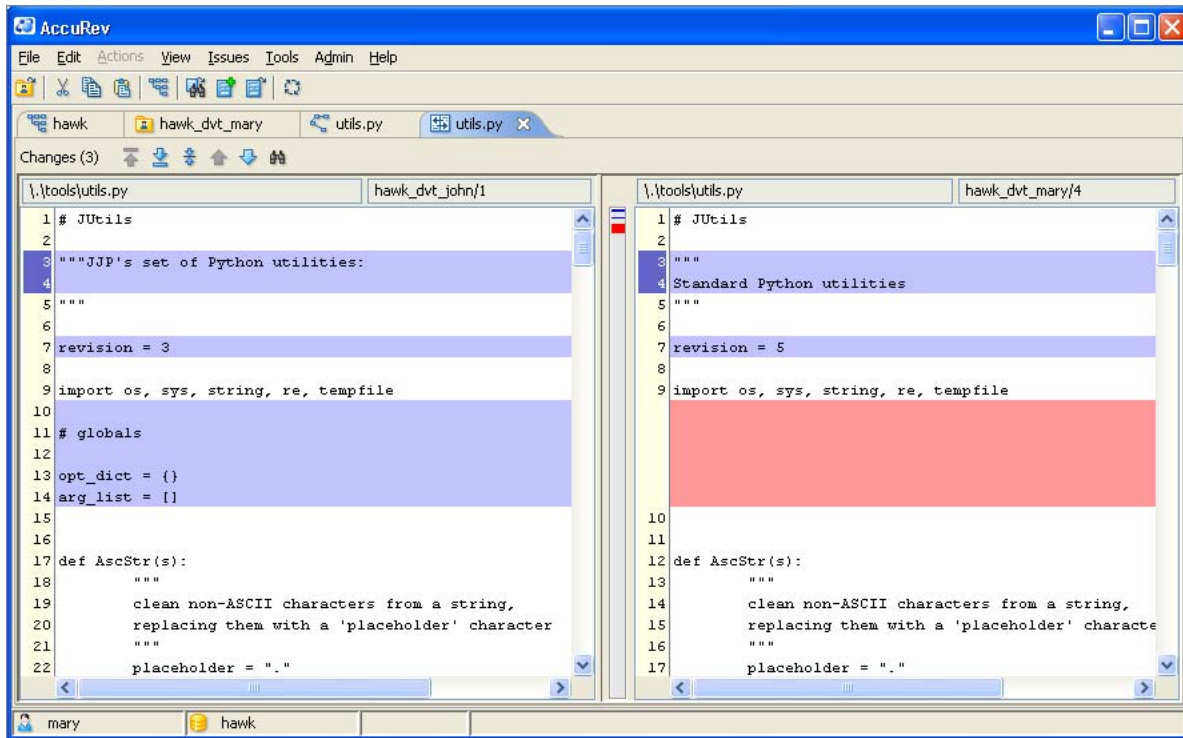
## Comparing Versions

You can compare any two versions of an element:

1. Right-click a version in the Version Browser display, and select **Diff Against Other Version** from the context menu. The mouse pointer changes, with a Diff-tool icon  annotating the standard arrow.
2. Left-click on any other version.



A new tab opens, in which the AccuRev Diff tool displays the two files side-by-side, highlighting their differences:




(This is the default; you can configure the Diff tool to invoke any file-difference program.) For more on the Diff tool, see *The AccuRev Diff, Merge, and Patch Tools* on page 115.

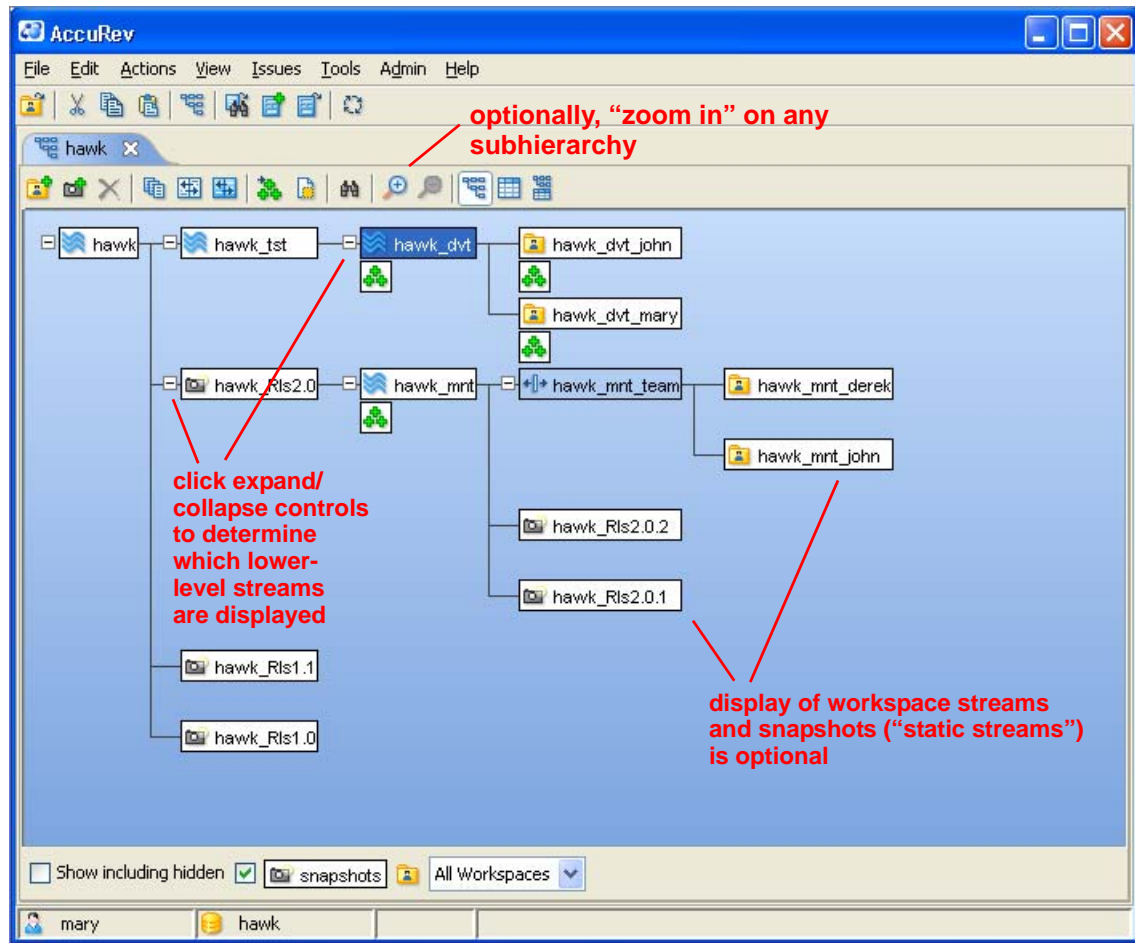
## Advanced Operations

This chapter has focused on the day-to-day AccuRev tasks that are most likely to be performed by developers. The GUI makes it equally easy for project leaders (or release engineers, or quality-engineering specialists) to get their work done. We'll take a look at just one of AccuRev's advanced configuration management features.

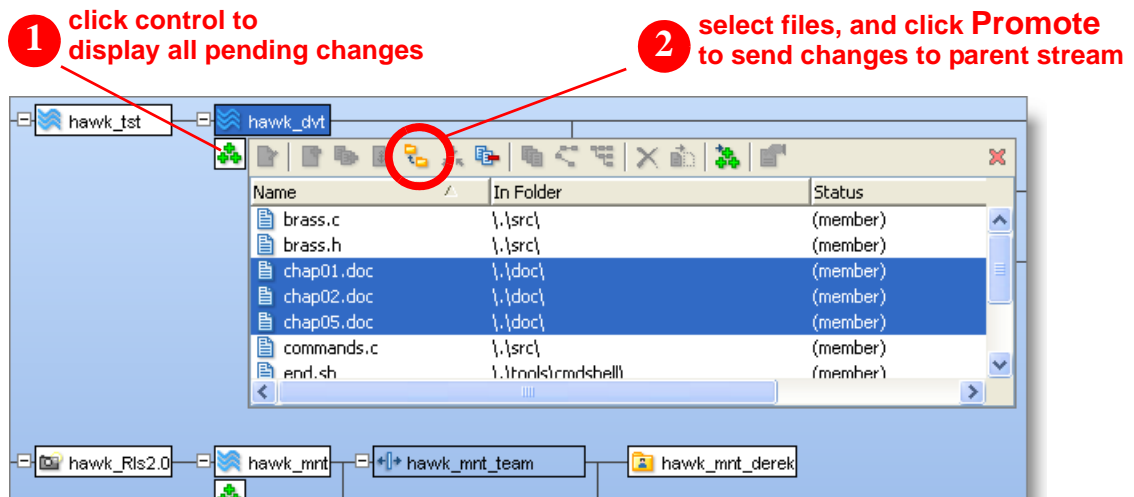
### Migrating Changes from Stream to Stream

AccuRev organizes all development of version-controlled files into streams. We've discussed the backing stream associated with developers' workspaces. More generally, a depot can have a complex hierarchy of streams, each one representing a separate development effort. There might be streams for Releases 3.2, 3.1.5, and 4.0 of a product, all active concurrently. In addition, there may be streams for special or experimental projects, for foreign-language translations, for emergency bugfixes, and so on. You can view, and work with, all of a depot's streams using AccuRev's StreamBrowser. To open a StreamBrowser tab, use the **View > Streams** command or the  **View Streams** toolbar button.





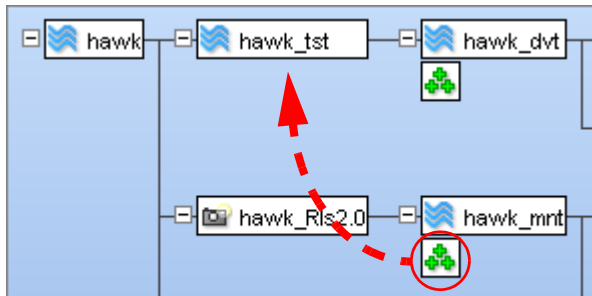
*Making Your Changes Public* on page 36 describes how a developer uses the **Promote** command to send changes from his own workspace to the backing stream. In a deep stream hierarchy, several promotions are required to propagate the changes all the way to the top. Each such promotion is simple.





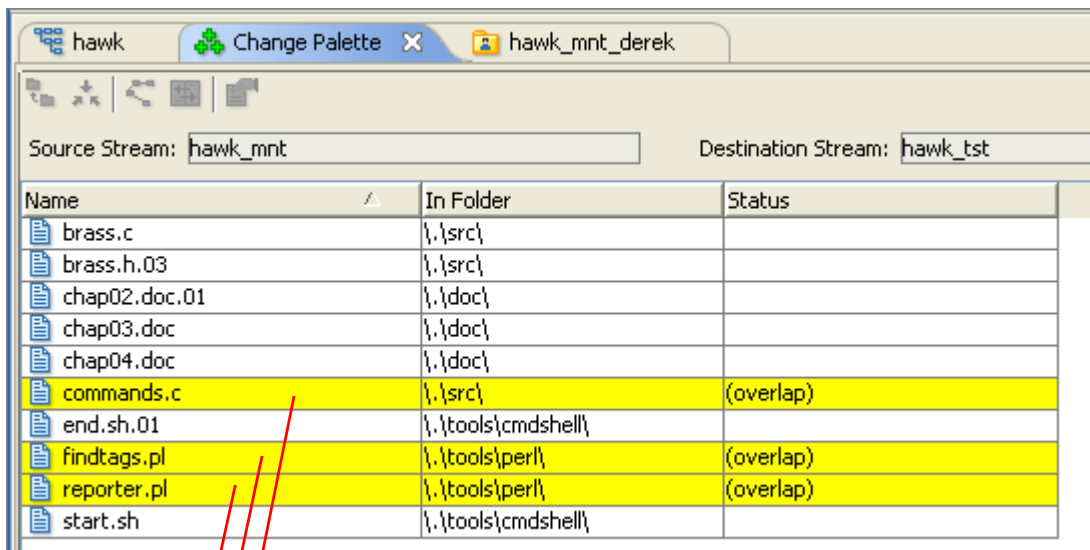
In many cases, promoting changes is even simpler than this — a single drag-and-drop operation does the trick!

It is often desirable to send changes to other streams, as well. For example, a manager might mandate that a developer's fix for a security hole be incorporated immediately into all active development efforts — i.e. into all active streams. The Change Palette makes it easy to migrate changes between streams, regardless of their location in the depot's stream hierarchy. This includes automatic determination of what changes need to be migrated, and control over the performing of any necessary conflict resolutions (merges). The following “storyboard” shows how to propagate changes from stream **hawk\_mnt** to stream **hawk\_tst**:



**1** drag-and-drop the pending-changes icon of the “source” stream to the “destination” stream

**2** Change Palette tab appears



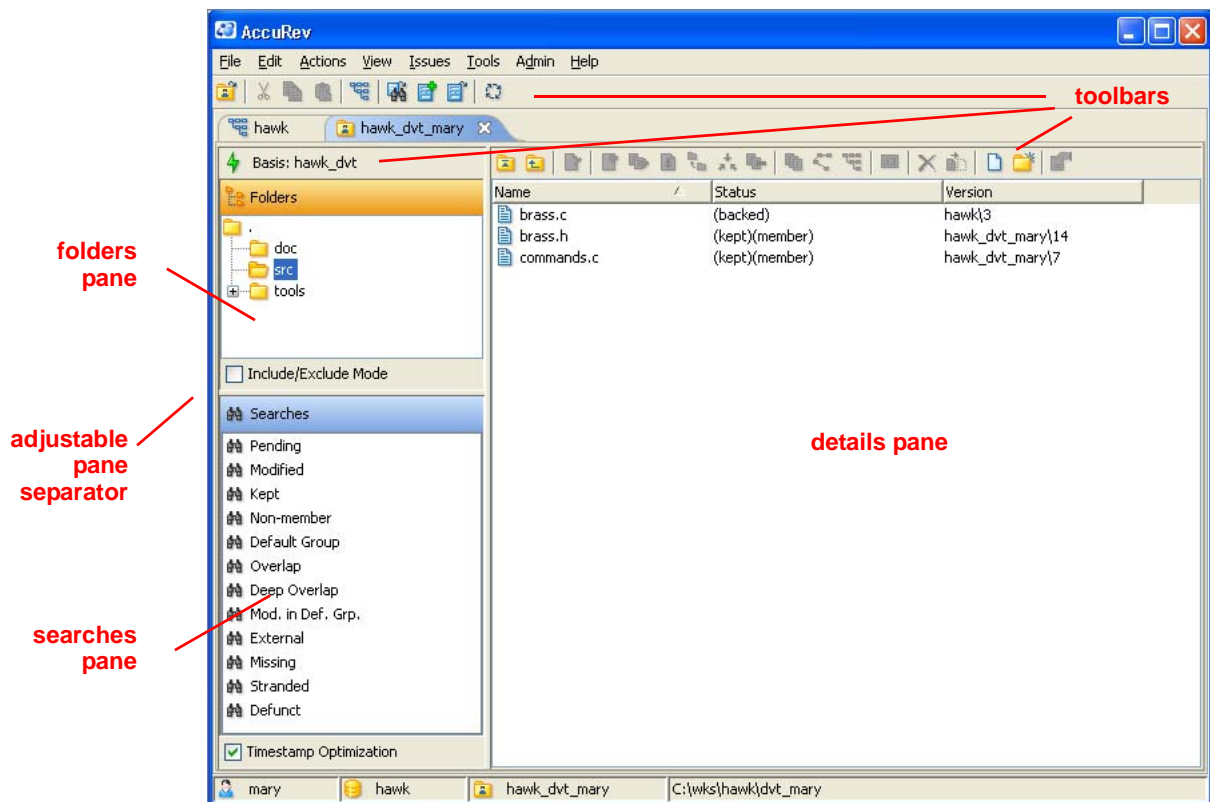
**3** these three versions must be merged before they can be promoted to **hawk\_tst** stream

all other versions can be promoted to **hawk\_tst** immediately

# The File Browser

AccuRev's job is to keep track of your files. Accordingly, one of AccuRev's main GUI tools is the File Browser. The File Browser makes it easy to view, monitor the status, and change the contents of files located in AccuRev workspaces. You can have any number of File Browser tabs open concurrently in an AccuRev GUI window — each one displaying the contents of a difference workspace.

The File Browser display resembles that of Windows Explorer. To the familiar folders pane and details pane, adds a unique searches pane. You can execute AccuRev commands by selecting from the GUI window's main menu, by clicking toolbar buttons, or by using the context (right-click) menus of items in the File Browser display.



As shown above, the folders pane enables you to navigate a workspace's folder (directory) hierarchy. When you're using the folders pane, the details pane displays the contents of the currently-selected folder.

The searches pane lists file-status searches that you can apply to the entire workspace. When you're using the searches pane, the details pane displays all the files, throughout the entire workspace, that meet the selected search criterion. For example, it can display all the files you've edited and saved with the **Keep** command. (In the AccuRev CLI, this facility is described using the term "filter".)

## Alternatives to the File Browser

The File Browser is not intended to completely replace the Windows Explorer or comparable tools on Unix systems. For example, the File Browser's details pane does not include such columns as Size, Date Modified, and Attribute. These are operating-system-level file properties; the File Browser reports only a few AccuRev-specific properties.


Some people do most of their work inside an integrated development environment (IDE), such as Visual Studio or Eclipse. These environments have their own “explorer” or “file browser” built in, typically organizing files into separate “projects”. You can use AccuRev commands within an IDE if an IDE integration for that environment is available, either from AccuRev, Inc. or from a third-party developer. Even if you use an IDE most of the time, you may want to use the File Browser occasionally:

- The IDE does not display all of a file's configuration-management properties, such as the current version-ID.
- Certain AccuRev commands cannot be executed through the IDE integration, only from the AccuRev File Browser.

It's important to keep in mind that you can use a wide variety of tools to make content changes to a file, but you must use AccuRev commands to make valid namespace changes: renaming, moving to another directory, or deleting. Some of the IDE integrations support AccuRev-level renaming.

## Opening a File Browser Tab

There are several ways to open a File Browser tab in the AccuRev GUI. When you start a new GUI session, one or more File Browser tabs (and other tabs) might open automatically, enabling you to continue from where you ended the previous session. To open a File Browser on any of your workspaces, use one of these techniques:


- Select **File > Open Workspace** from the command menu (or click the  **Open Workspace** toolbar button). Select a workspace and click **Ok**, or double-click it.
- On a StreamBrowser tab, make sure the desired workspace is displayed. To open another user's workspace, change the listbox setting at the bottom on the tab from **Current User** to the appropriate username. There is also an **All Workspaces** setting. Right-click the desired workspace, and select **Open** from the context menu. Or just double-click the desired workspace.



A new File Browser tab opens with the top-level folder in the folders pane selected. See the descriptions of the folders and searches panes under *File Browser Layout* on page 45.

## Leaving a File Browser Tab

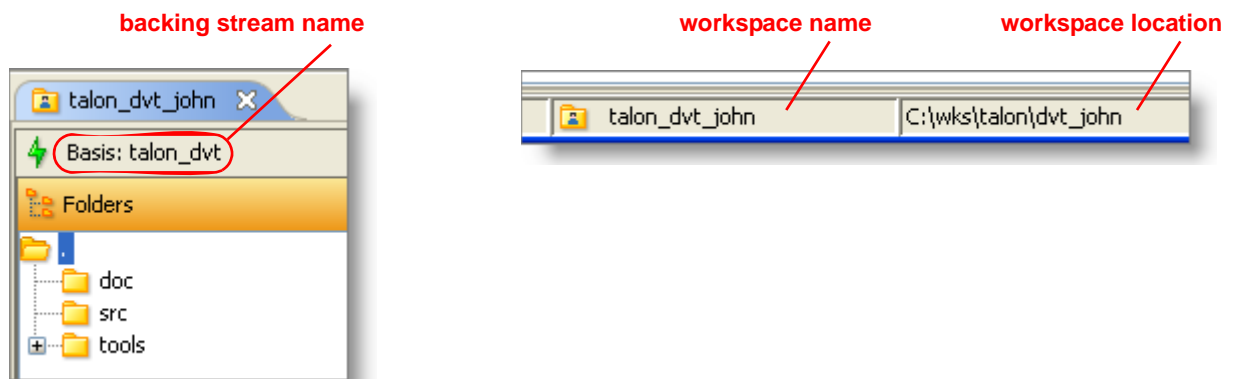
You can leave a File Browser, switching to another tab in the AccuRev GUI, then return to that File Browser tab later. When returning to a File Browser tab, it's a good idea to refresh the display

(**View > Refresh**, the  toolbar button, or function key **F5**). This ensures that the File Browser display reflects any work you performed “between visits” to the tab.

Alternatively, you can close the tab altogether: right-click the title, then select **Close** from the context menu. If AccuRev Look And Feel is enabled (**Tools > Preferences**), you can close the tab using the “X” icon on the tab control itself.

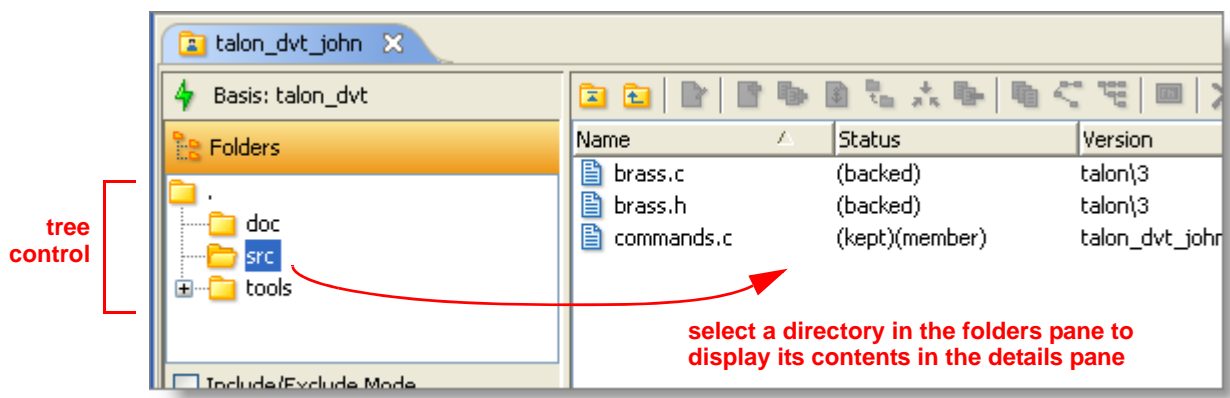
## File Browser Layout

If you’re familiar with Windows Explorer or similar tools, you’ll find that using the File Browser is easy. As with those tools, there’s a folders pane (navigation pane) and a details pane. AccuRev adds a third searches pane. When you’re working in the File Browser, the indicators at the bottom of the AccuRev GUI window show the name and location of the current workspace. The name of the workspace’s backing stream (also called the basis stream or parent stream) is listed above the folders pane.



## Folders Pane

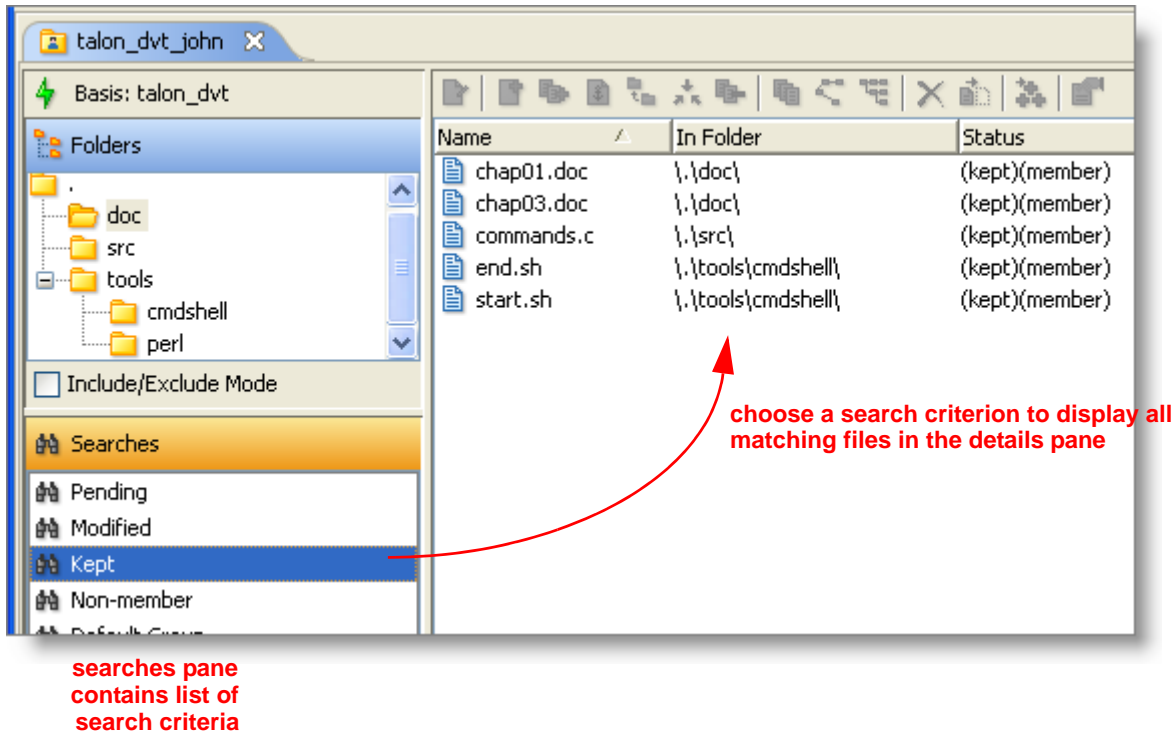
The folders pane provides for “traditional” navigation: when you select a particular folder (directory) using the standard tree control, the files in that directory appear in the details pane.



## Searches Pane

The File Browser can also organize the workspace’s files by AccuRev status, instead of by directory location. When you click one of the items in the searches pane (e.g. **Kept**), AccuRev

searches the entire workspace for elements that meet that search criterion, and displays those elements in the details pane:



The searches view enables you to see “just the files you care about”, instead of all the files in your workspace. Which files do you care about? Roughly speaking, a workspace contains a copy of its backing stream — often, the entire source base — which might include hundreds or thousands of files. But for a given development project, you’ll probably modify only a handful of the files. You may not need the other files at all; or you may need them for general reference, or to enable you to perform software builds and tests in the workspace.

Note: there’s another way to concentrate on just the files you care about: the Include/Exclude facility. See *Working in Include/Exclude Mode* on page 67.

The following section discusses the details pane and the various AccuRev file-status indicators. In section *AccuRev File Statuses* on page 48, we discuss the various search criteria that you can choose in the searches view.

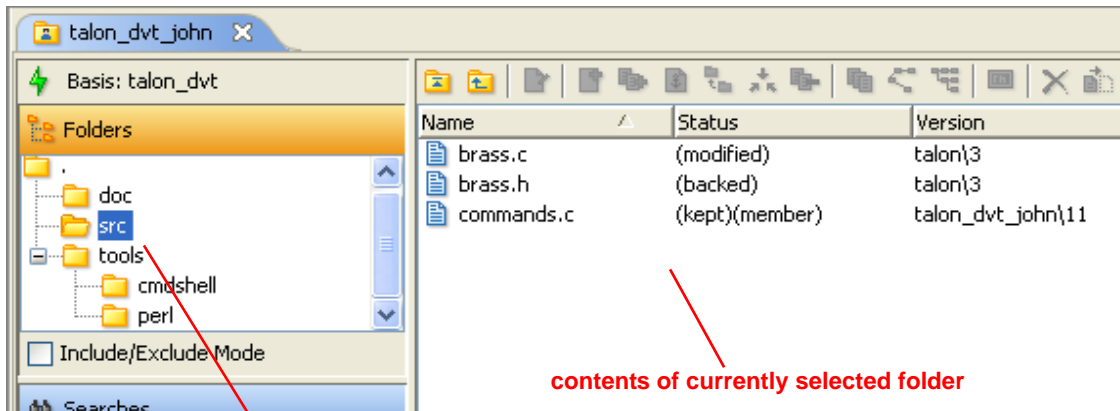
## Details Pane

The details pane of the File Browser displays several columns of information for the files and directories in a workspace:

### Name

The name of the object, either a file or a subdirectory, in this workspace. AccuRev allows you to rename and relocate objects, so a file might have a different name and/or a different directory location in another workspace.

When the details pane is displaying the contents of a single folder, the **Name** column shows the name of each object in the folder. The highlight in the folders pane indicates which folder is being displayed.



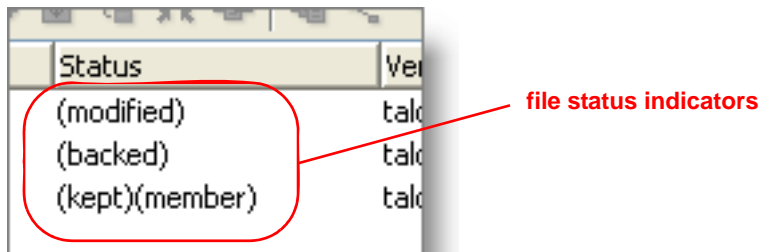
currently selected folder

contents of currently selected folder

When the details pane is displaying the results of a whole-workspace search, a user preference controls whether element pathnames are displayed as a single column (**Element**) or two columns (**Name** and **In Folder**). See [User Preferences](#) on page 6.

## Status

One or more keywords, indicating the AccuRev status of the element in this workspace. [AccuRev File Statuses](#) on page 48 describes the file statuses. [Common Usage Scenarios](#) on page 51 includes notes on how the status of an element changes during typical development scenarios.



file status indicators

## Version

A version-ID, in the form of a stream name followed by an integer version number. The version-ID indicates which version of the element is in this workspace. For elements that you have modified in this workspace, but not yet promoted to the backing stream, the version-ID indicates a version in the workspace stream — a special stream that is dedicated to this workspace. These are described as “active elements” or in AccuRev parlance, as “members of the workspace’s default group”. For inactive elements (those you haven’t modified or [Anchor](#)’ed), the Version column indicates a version from some higher-level stream, which your workspace stream inherits.

Version	
talon\3	— “inactive” element: version inherited from higher-level stream
talon\3	— “inactive” element: version inherited from higher-level stream
talon_dvt_john\11	— “active” element: version in workspace stream

If this column is blank, the file or subdirectory has not been placed under version control. Such objects have the status (**external**).

## AccuRev File Statuses

Development work involves making changes to elements — both files and directories. As files get edited in workspaces, new versions get created in workspace streams, and existing versions get promoted to higher-level streams, AccuRev shows you the current status of each element. Keep in mind that the status of an element in your workspace may be different from its status in other workspaces and streams.

An element’s file status is indicated by a set of one or more status indicators (or flags), each in the form of a parenthesized word. For example:

`(overlap)(kept)(member)`

Even though we use the term “file status”, each directory element also has a status in each workspace and stream. A subset of the status indicators is used to report directory statuses.

Here are the AccuRev status indicators, organized by category:

*Presence of the element in the workspace:*

- **(defunct)** — the element has been marked for removal from the workspace stream with the **Defunct** command. The element has already been removed from the workspace tree (local disk storage); it gets removed from the workspace stream (in the depot) when you **Promote** the element to the backing stream.
- **(external)** — the file or directory has not been placed under version control. (It’s in the workspace tree, but not in the workspace stream.) See *Controlling the Display of External Objects* on page 50.
- **(excluded)** — the element does not appear in the workspace because it has been excluded, using the Include/Exclude facility. For file elements, it’s more likely that the exclusion was explicitly set on the directory in which the file resides, or in a higher-level directory that includes the file. See *Working in Include/Exclude Mode* on page 67.
- **(link)** — (Unix/Linux link-based workspace only) the element is represented by a symbolic link to an element in the associated reference tree. This is the state of elements that are not currently under active development (not in the workspace’s default group).

- **(missing)** — the workspace “should” include a version of this element, but doesn’t. This occurs when you delete version-controlled files from the workspace tree using operating system commands, or using the AccuRev **Delete** command.

In a sparse workspace, this status also applies to elements that are not currently loaded into the workspace. The ability to create a new sparse workspace was disabled in AccuRev Version 3.5; use the Include/Exclude facility in a new workspace instead (see page 67).

- **(stranded)** — the element is active in the workspace, but there currently is no pathname to the element. See *Version Control of Namespace-Related Changes* on page 51 of *AccuRev Technical Notes*.

*Changes to the element in the workspace:*

- **(modified)** — the file has been modified in the workspace since the most recent **Update** or **Keep**. See *Controlling the Determination of (modified) Status* on page 50.
- **(kept)** — a new version of the element has been created with **Keep**, **Rename**, or **Defunct** (or the CLI command **undefunct**) and the file has not subsequently been modified, promoted to the backing stream, or purged (**Revert to Backed**).
- **(member)** — the element is “active” in the workspace (is in the workspace stream’s default group). The commands that create a new version, listed above, also make the element active. So do the commands **Anchor** and **Send to Workspace**, and the CLI command **revert**.

*Relationship to the version in the backing stream:*

- **(backed)** — the version in the workspace stream is the same as the version in the backing stream. And you have not changed the element since the last time you **Promote**’d it or purged it with **Revert to Backed**, or since the most recent **Update** of your workspace.
- **(stale)** — the element needs to be updated, because the version in the backing stream has changed since the workspace’s latest **Update**. And since you have not changed the element in your workspace, it can be safely updated.
- **(overlap)** — the element has changed both in the backing stream and in your workspace. This indicates that a merge is required before you can promote your changes to the backing stream. Elements with this status appear with a yellow highlight.

The following sections describe ways in which you can control how the File Browser determines file statuses. And see *Appendix: File Statuses and Searches* on page 85 for a “grand unified theory” discussion and pictorial representation of the file statuses.



## Controlling the Display of External Objects

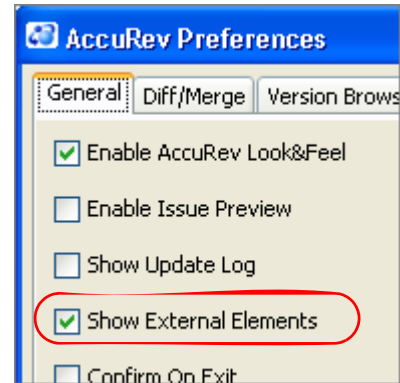
The details pane can show both elements — files and subdirectories that have been placed under version control — and external objects. The **Show External Elements** preference provides an all-or-none control over the display of external objects.

Note: in previous releases, this preference was implemented through a checkbox in the File Browser itself.

Use the command **Tools > Preferences** to open the AccuRev Preferences window.

With a little more work, you can have the details pane display some, but not all, external objects. You set the **Show External Elements** preference, but suppress the display of selected objects through environment variable ACCUREV\_IGNORE\_ELEMS. For more on this facility, see *Pathname Optimization* on page 84.

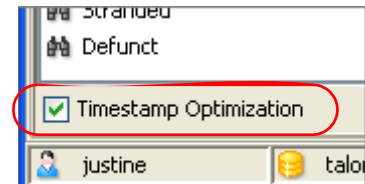
**all-or-none  
control over  
the display  
of external  
objects**



## Controlling the Determination of (modified) Status

Before displaying the contents of a folder in the details pane, AccuRev determines the status of each object (file or subfolder) in the folder. When you work in the searches pane, AccuRev determines the status of each object in the entire workspace. To determine whether a file element's status should include the **(modified)** indicator, AccuRev checksums the file and compares the result with the checksum of the version in the workspace stream.

If the **Timestamp Optimization** checkbox (at the bottom on the File Browser tab) is checked, AccuRev skips files whose timestamps precede the last time the workspace was scanned for modified files (for example, by **Update**). For more on this facility, see *Timestamp Optimization* on page 82.



## Working in the Details Pane

Working with a workspace's files in the File Browser's details pane follows this pattern:

1. Select one or more files and/or directories.
2. Invoke an AccuRev command, either:
  - ... by clicking one of the File Browser's toolbar buttons, or
  - ... by right-clicking a selected object and choosing a command from the context menu, or
  - ... by choosing a command from the **Action** submenu of the GUI window's main menu.

The file-selection gestures are the same ones used by Windows Explorer:

- To select a file, click it with the left mouse button.

- To select a contiguous range of files, click-and-drag with the left mouse button.
- To select all the files in a directory, select any one of them, then press **Ctrl-A**.
- To add or subtract a file from an existing selection, hold down the **Ctrl** key and click that file with the left mouse button.
- To extend an existing selection to a particular file, hold down the **Shift** key and click that file with the left mouse button.
- Typing a character selects the next file or directory whose name begins with that character (if the details pane listing is sorted on the Name column).

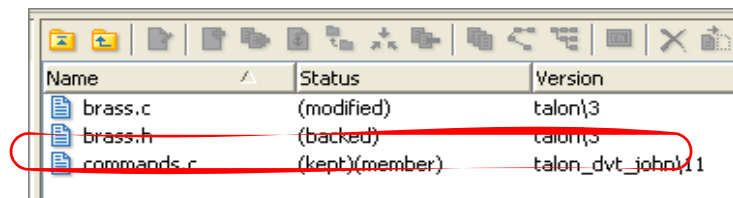
Next, we present common scenarios for working with elements listed in the File Browser's details pane. This is followed by a reference to the commands that you can invoke on the elements.


## Common Usage Scenarios

As a version control system, AccuRev keeps track of changes that users make to files and directories. The following sections describe common usage scenarios, showing how AccuRev change-tracking is reflected in the File Browser's details pane: in changes to the status of objects, and in creation of new versions of objects.

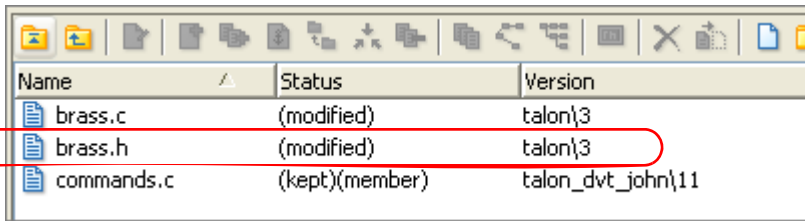
### Editing a File's Contents

Perhaps the most common scenario is “non-conflicting development”: modifying a file that no one else is working on concurrently. Initially, the Status column in the details pane shows the file's status as **(backed)**: your workspace contains an unmodified copy of a version in the backing stream. The Version column contains a version-ID, indicating which backing-stream version it is.




Invoke the  **Edit** command on the file, using the toolbar button or the file's context menu. This launches a text editor session on the file. You can use environment variable AC\_EDITOR\_GUI or EDITOR to control which text editor is launched.

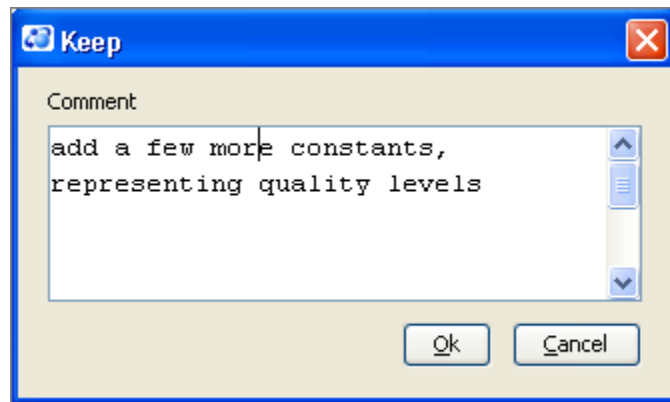
When you end the edit session, the file's status changes to **(modified)**. This indicates that you've modified the file in your workspace since the last time you synchronized the file in your workspace with the depot, for example with a **Keep** or **Update** command. (That is, you've changed the file in the workspace tree, but not the element in the AccuRev repository.)



Name	Status	Version
brass.c	(modified)	talon\3
brass.h	(modified)	talon\3
commands.c	(kept)(member)	talon_dvt_john\11

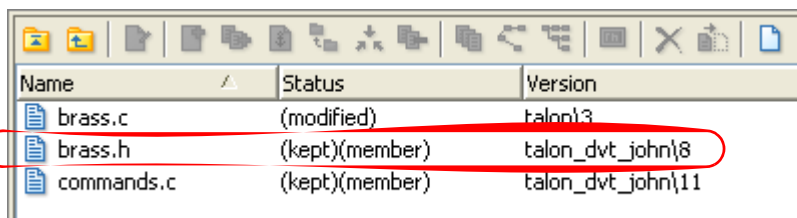
You can edit the file (and update the File Browser display) as many times as you wish. The file's status remains **(modified)**.

To have AccuRev record a new version of the file, invoke the  **Keep** command on it. If you wish, enter a comment in the dialog box that appears. The comment string becomes a permanent annotation to the version, viewable with the History Browser.





The **Keep** command creates a new version of the file in the workspace stream. This version is said to record a content change to the file. AccuRev also tracks namespace changes — see below.

The file's status changes to **(kept)(member)**; this indicates that you've recorded a new version, and that the file is currently active in your workspace (is a member of the workspace's default group).






Name	Status	Version
brass.c	(modified)	talon\3
brass.h	(kept)(member)	talon_dvt_john\8
commands.c	(kept)(member)	talon_dvt_john\11

The Version column shows the version-ID of the newly created version. Note that this version is recorded in your private workspace stream (in this example, **talon\_dvt\_john**); previously the Version column indicated that your workspace contained a version from the public backing stream (**talon**).

You can continue modifying the file with the  **Edit** command, and saving new versions in the depot with the  **Keep** command. The file's status will alternate between **(modified)(member)** and **(kept)(member)**. The persistence of the **(member)** indicator reflects the fact that the file remains active in your workspace until you promote your changes to the backing stream or undo your changes. These operations are described below.

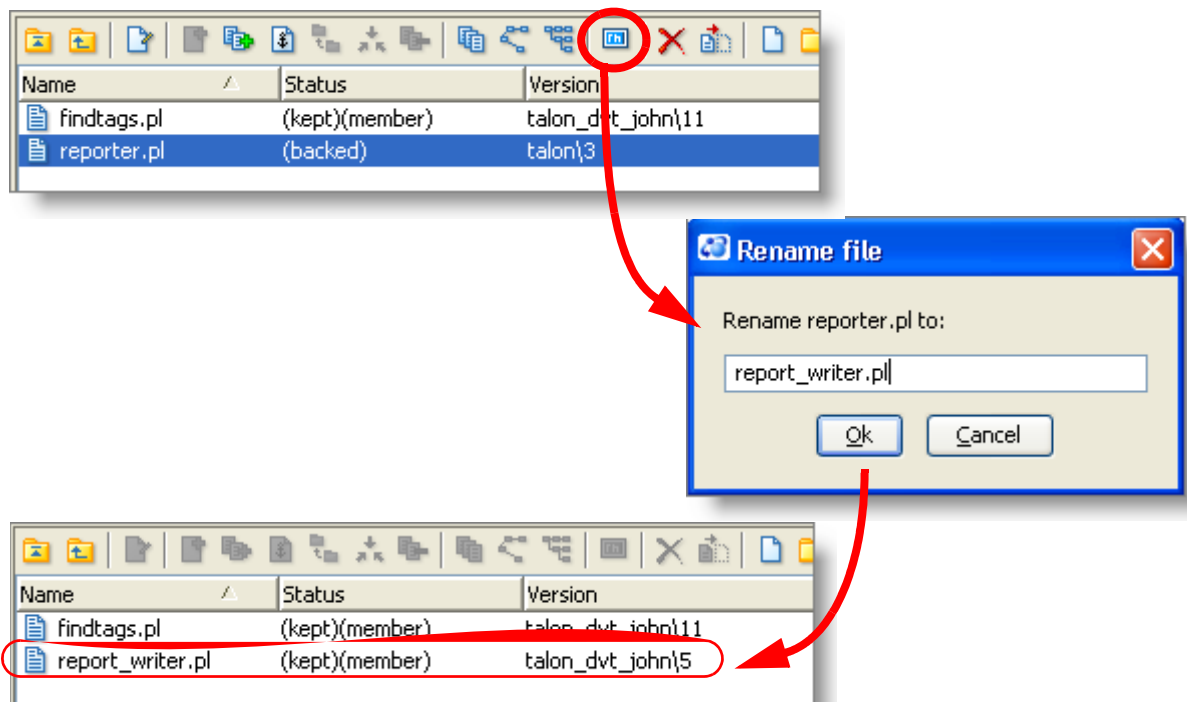
## Renaming or Moving a File

In addition to tracking changes to the contents of files, AccuRev tracks namespace changes:

- To rename an element within the same directory, invoke the  **Rename** command. (You can't use this command to rename (**external**) objects).
- To move a file to a different directory in the depot, right-click the file and select  **Cut** from the context menu. Then right-click the destination directory in the folders pane (not the details pane) and select  **Paste** from the context menu. You can cut-and-paste a multiple-file selection in the same way.

You can also use the **Rename** command to move a file to a different directory; specify a relative pathname (such as `..\otherdir\myfile.c`) or a depot-relative pathname (such as `./lib/header/base.h`) as the new name for the file.

AccuRev records a namespace change to a file in the same way it records a content change: by creating a new version of the file in the workspace stream.

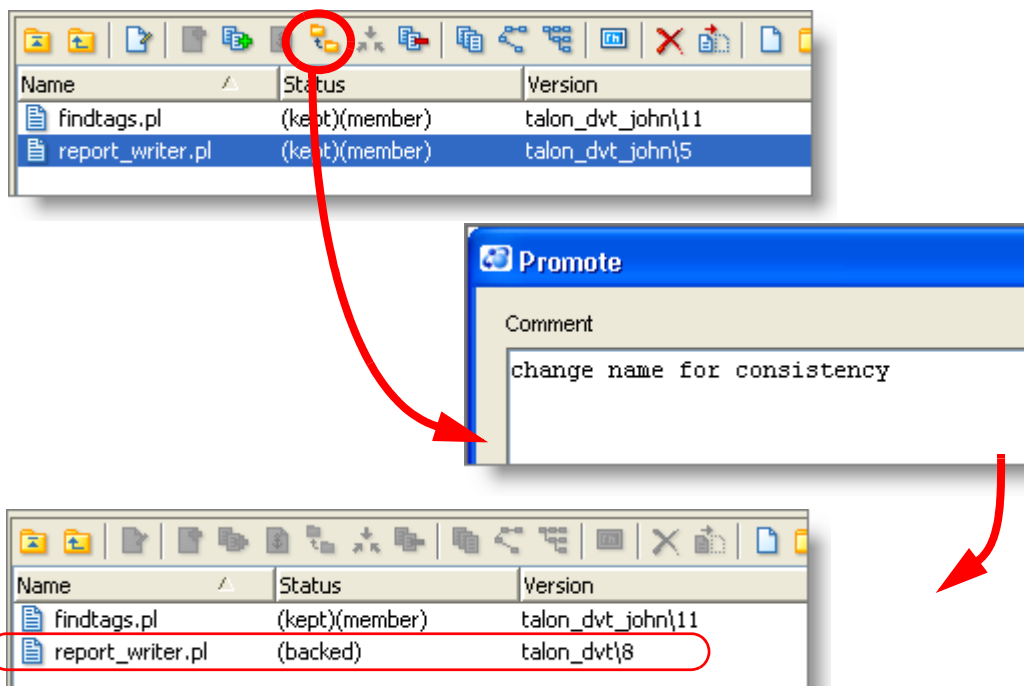


As with a content change, the file's status changes to **(kept)(member)**. (If the file also has content changes that have not yet been saved with **Keep**, the status becomes **(modified)(member)**). Note that making a namespace change to a file “activates” it — creates a new version in the workspace stream and makes it a member of the workspace's default group — just like **Keep**'ing a content change.

You can also rename and/or move a directory — see [Changing a Directory](#) on page 55.

## Following Through by Promoting the Changes

Initially, the content changes and/or namespace changes you make to a file are recorded only in your workspace's private stream. This keeps your work isolated from your colleagues' work. When you're ready to share your changes to a file with your colleagues, you **Promote** the active version from your workspace stream to the backing stream. This makes your changes available to all workspaces that are based on the same backing stream.




Note how the version-ID in the Version column changes:

- Before the promotion, it indicates the particular version of the file that is active in your workspace (in this example, version **talon\_dvt\_john\5**).
- After the promotion, it indicates that version's newly created version-ID in the backing stream (in this example, **talon\_dvt\8**).

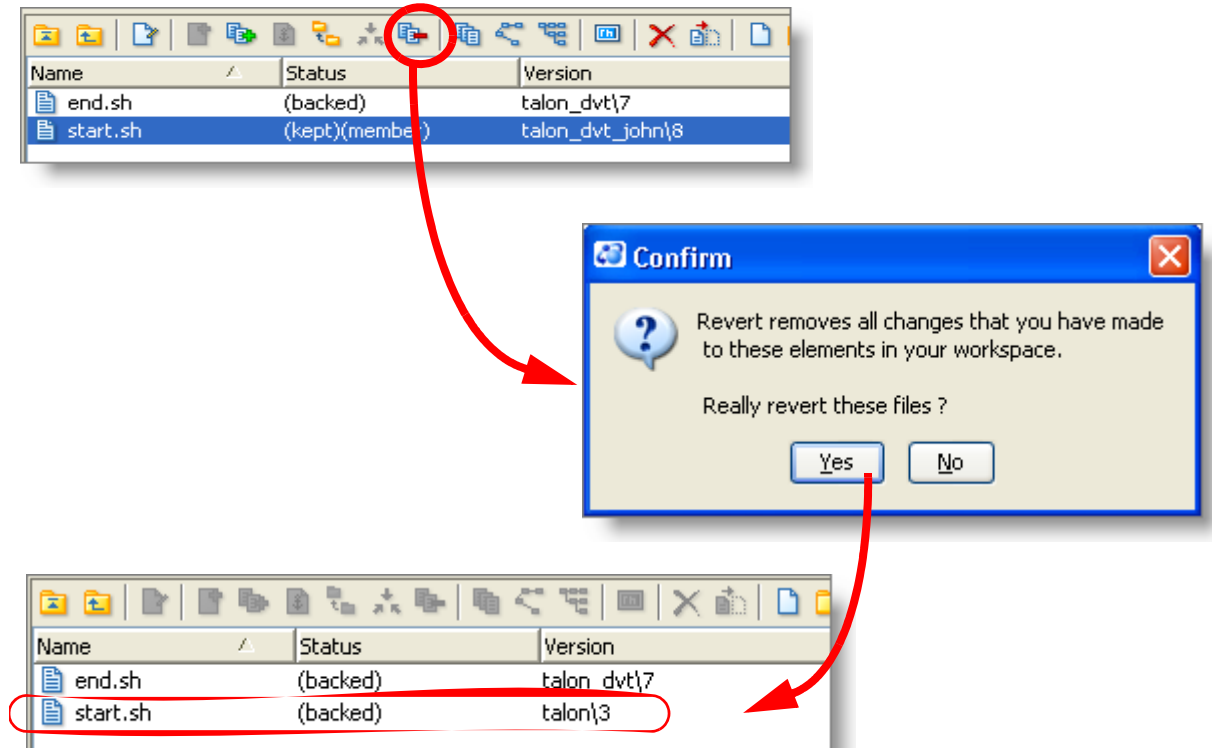
Having been promoted, the file is no longer active in the workspace (is no longer in the default group of the workspace stream). Accordingly, it loses its **(member)** status and returns to being **(backed)** — its original status before you started working on the file. That is, the workspace returns to “inheriting” the version of the file currently in the backing stream — which happens to be the version you just promoted there!

Note: strictly speaking, you must **Keep** a file's changes before you can **Promote** them. But as a convenience, the File Browser enables the **Promote** command for files that are **(modified)**, but not **(kept)**. Invoking **Promote** performs both a keep transaction and a promote transaction.

## Following Through by Undoing the Changes

Inevitably, you sometimes decide *not* to share your changes to a file with your colleagues — instead, you decide to discard the changes altogether. The  **Revert to Backed** command

undoes all the content and/or namespace changes you've made to an active file. The file's status reverts to **(backed)**, and your workspace "rolls back" to using the version that it contained the last time the file's status was **(backed)**. It might be a version that you brought into your workspace with a recent **Update** command; or it might be a version that you created in your workspace, then **Promoted** to the backing stream.



Note: if your changes to a file included moving it to a different directory, invoking **Revert to Backed** causes the file to disappear from its new location and return to its original location.

A file's context menu (but not the File Browser's toolbar) also contains a variant command, **Revert to Most Recent Version**. This command is useful if you've modified a file's contents repeatedly, creating one or more intermediate versions in your workspace with the **Keep** command. **Revert to Most Recent Version** discards any content changes you've made since the most recent **Keep**. The file's status reverts from **(modified)** to **(kept)**. The file remains active in the workspace, so it retains its **(member)** status.

### Changing a Directory

In addition to tracking changes to files, as discussed in the preceding sections, AccuRev tracks changes to directories. AccuRev's model for directory-level changes is simple, but somewhat different from the model used by the operating system (and by some other version-control systems). AccuRev considers the following to be changes to a directory:

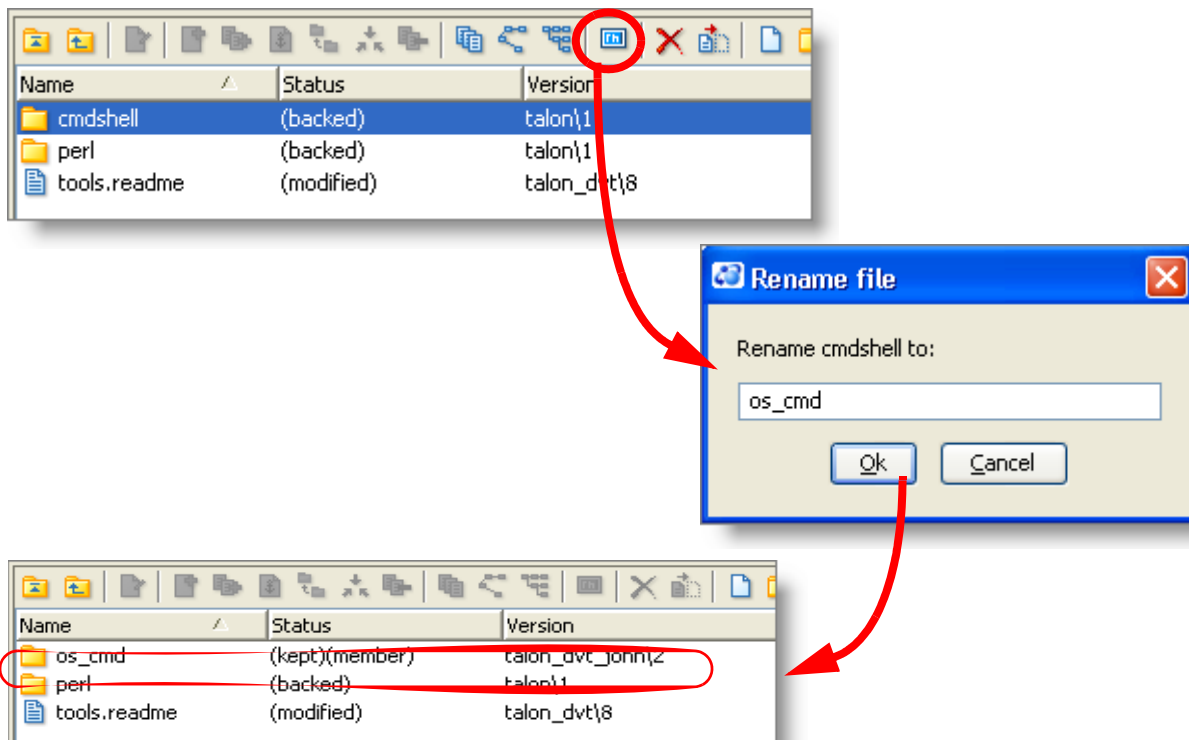
- Renaming a directory
- Moving a directory to another location in the depot's directory hierarchy
- Deleting a directory

The following are *not* changes to a directory:

- Creating a new file (it's a change to the file itself)
- Renaming an existing file (it's a change to the file itself)
- Deleting a file (it's a change to the file itself; see *Deleting a File — Intentionally and Permanently* on page 59)

Note that it is only changes involving a directory's pathname that are considered to be changes to the directory itself. Changes to a directory's contents are not considered to be changes to the directory — they are changes to the affected elements within the directory.

You change a directory's pathname in the same way you change a file's pathname — with the **Rename** command or with **Cut** and **Paste** commands (see *Renaming or Moving a File* on page 53). When you make such a change, AccuRev records a new version of the directory in the workspace stream.



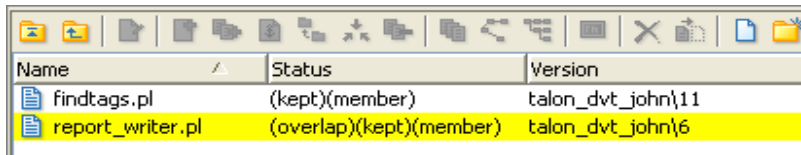
The new version-ID appears in the Version column of the details pane (in this example, **talon\_dvt\_john\2**). And the directory's status changes to **(kept)(member)** — just as it does for a file's namespace change.

## Merging Your Changes with Someone Else's Changes

AccuRev supports concurrent development: two or more users can start with the same version of a file and make changes to that file independently — both content changes and namespace changes. After one of the users **Promotes** his changes to the backing stream, each of the others must **Merge** her own changes with the newest version in the backing stream. Merging is

described in chapter *The AccuRev Diff, Merge, and Patch Tools* on page 115. We describe here how a merge scenario is reflected in the File Browser display.

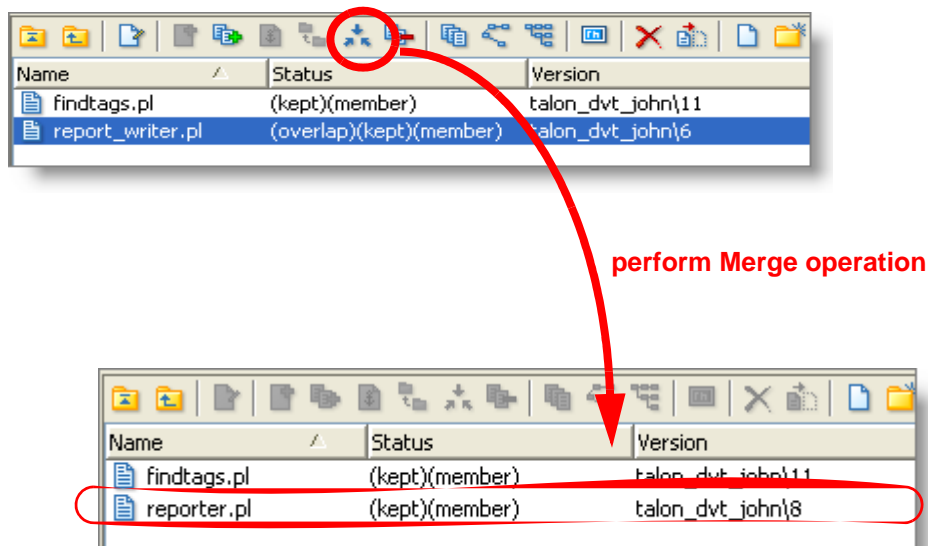
As you work on a file, **Keeping** intermediate versions in your workspace, the file's status alternates between **(modified)** and **(kept)**. At any point, you may notice that an additional indicator, **(overlap)**, appears in the Status column. To make sure you notice, the File Browser displays the entry with a yellow highlight.



Name	Status	Version
findtags.pl	(kept)(member)	talon_dvt_john\11
report_writer.pl	(overlap)(kept)(member)	talon_dvt_john\6

This means that a new version of the file has entered the backing stream. Typically, one of your colleagues has edited the file and promoted her version to the backing stream. It may also be that someone has promoted a new version of the file to a higher-level stream, and the backing stream dynamically inherits the new version “from on high”.

Whenever a file's status is **(overlap)**, the **Merge** command is enabled. The execution of a **Merge** command concludes with the **Keeping** of a new version in your workspace.



(In this example, the **Merge** involved incorporating a namespace change: the file's name has changed from **report\_writer.pl** to **reporter.pl**.)

When you **Promote** this merged version to the backing stream, the file's status returns to **(backed)**. This step is exactly the same as in section *Following Through by Promoting the Changes* on page 54.

### Deleting a File — Accidentally or Temporarily

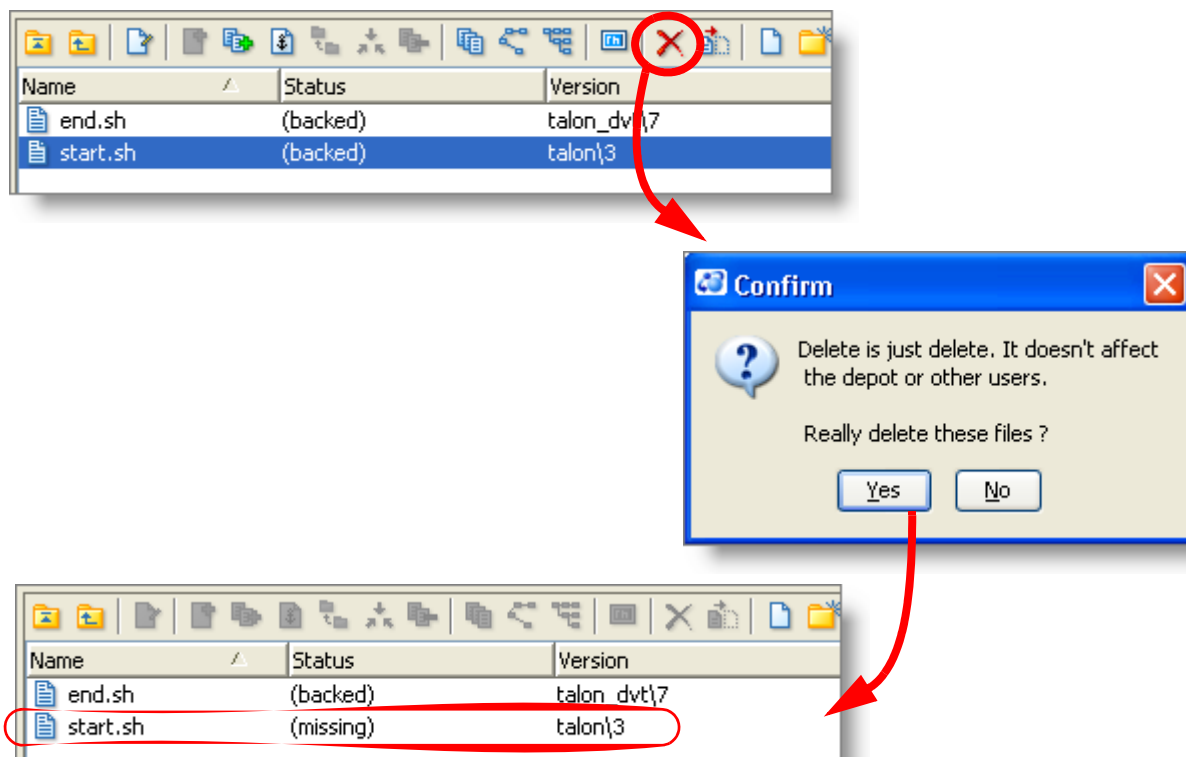
An AccuRev workspace is an ordinary directory tree (the workspace tree), typically located on your machine's hard drive. Nothing prevents you (or perhaps, some rogue cleanup script) from



using operating system commands to delete one or more of the files under version control. On occasion, you may even want to delete some files temporarily — for example, to test the robustness of your build or installation procedure.

By definition, deleting a file at the operating system level makes it disappear from disk storage. Operating system tools, such as Windows Explorer or the Unix **ls** command, will detect that the file no longer exists. But the file does *not* disappear from the File Browser display. AccuRev knows that the file *should* be in the workspace, because the file element still exists in the workspace's built-in stream. (The workspace stream is located in the AccuRev depot. It's unaffected by the operating system's delete-file commands.)

Accordingly, when a version-controlled file is deleted at the operating system level, the File Browser continues to list it, but indicates the file's status as **(missing)**.



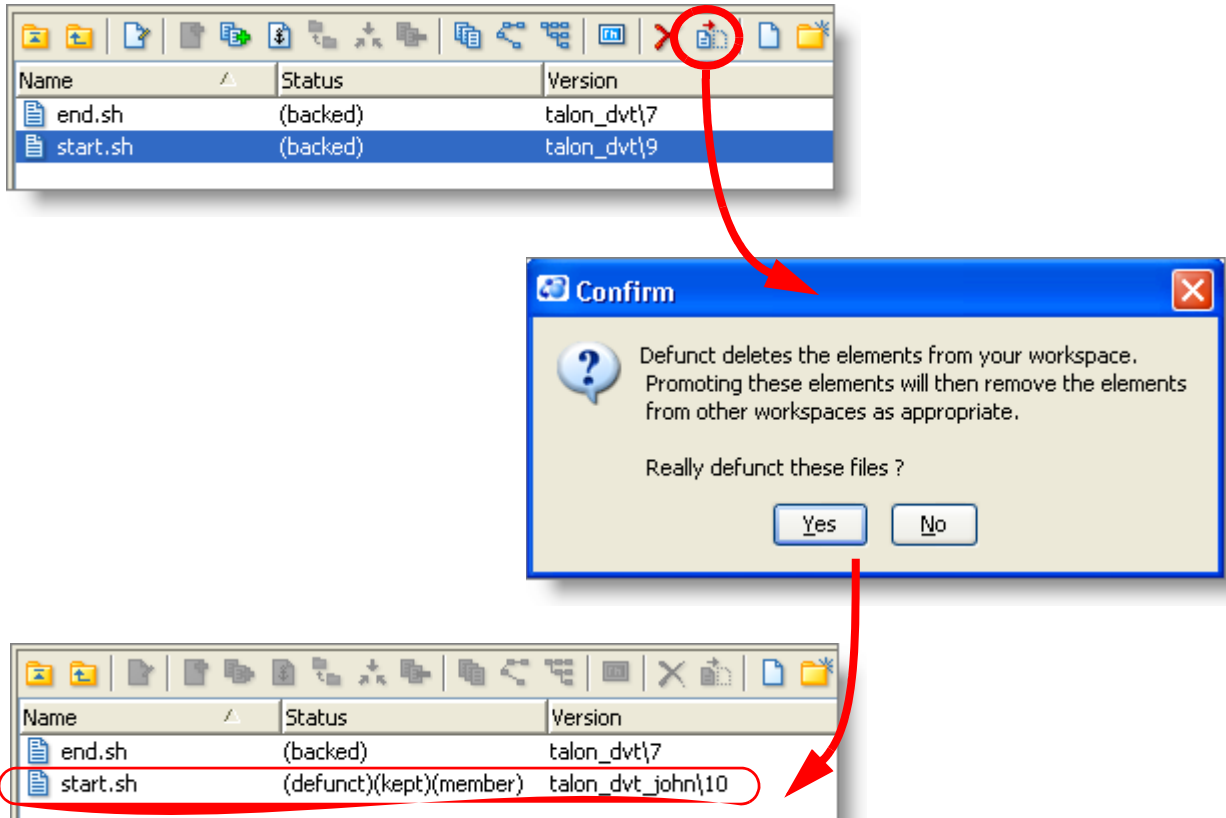
Note that the Version column continues to indicate which version of the file should be in the workspace. To restore that version, invoke the **Populate** command from the missing file's context menu.

Deleting a directory at the operating system level is similar to (but potentially more destructive than) deleting a file. The entire directory subtree is deleted from disk storage, and the directory is listed by the File Browser as **(missing)**. To recover the entire directory tree, invoke the **Populate** command from the directory's context menu. Be sure to select the Recursive option from the Populate Files dialog box.

## Deleting a File — Intentionally and Permanently

Sometimes, you want to delete a file permanently. That is, you want the file to disappear from your workspace, and from other users' workspaces, too. The file might be related to a product feature that was cancelled. Or perhaps a code reorganization rendered the file unnecessary.

Deleting a file at the AccuRev depot level (rather than simply at the operating system level) is called defuncting, and is implemented by the **Defunct** command.



Defuncting a file removes it from the workspace's disk storage (that is, deletes the file at the operating system level). In addition, the **Defunct** command is recorded in the depot. It may be surprising at first, but AccuRev manages the defuncting of a file in the same way as it manages the creation of new versions. To AccuRev, defuncting is just another kind of change that can happen to a file:

- Defuncting "activates" a file in the workspace stream, recording the fact that you've made a change to the file. In addition to getting **(defunct)** status, the file gets the **(kept)** and **(member)** statuses, just as if you had performed a **Keep** command.
- AccuRev records the change as a new version of the file in the workspace stream (in this example, version **talon\_dvt\_john\10**).
- At first, the defuncting of a file is isolated to your own workspace. The file continues to exist in other users' workspaces.

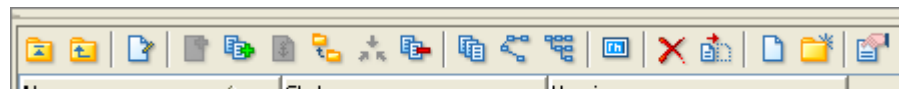
- To “share” the defuncting of a file, you **Promote** the change to the backing stream. This causes the file to disappear from your workspace stream, and from the File Browser display. The file will disappear from other users’ workspaces when they invoke the **Update** command.
- As always, you propagate the change — in this case, removal of the file — throughout the depot by promoting the defuncted file from the backing stream to the depot’s higher-level streams.

Note that the **Defunct** command does not actually remove the file element from the AccuRev repository, even if you promote the change to the depot’s base stream. Because AccuRev is TimeSafe, old versions of the file continue to exist in snapshots of streams, in History Browser displays, etc. You can restore such elements with the CLI command **undefunct**.

## File Browser Command Reference

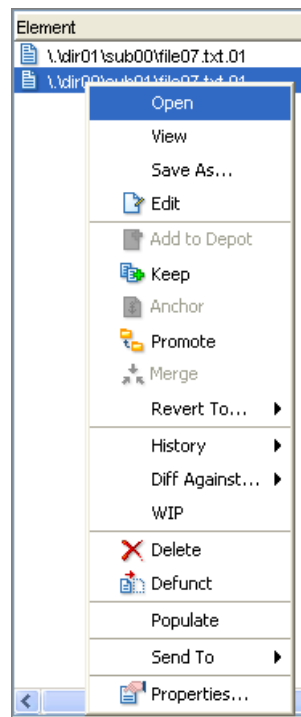
The File Browser commands listed below can be invoked in several ways:

- The toolbar above the details pane.
- The context (right-click) menu of an object.
- Under **Actions** on the GUI window’s main menu.

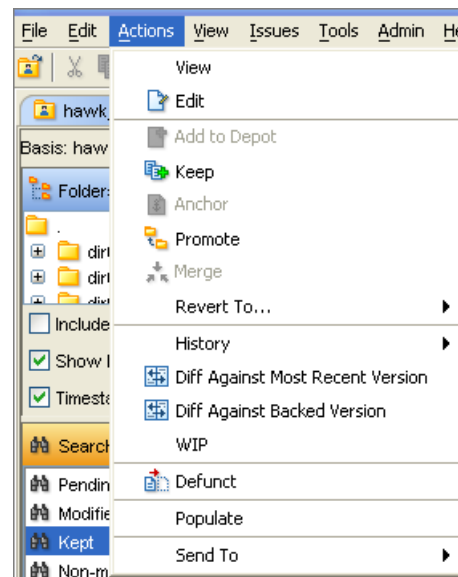


details pane toolbar

context menu of object



Actions menu



Some of the commands cannot be invoked in all these ways. And note that you invoke one of the most important commands, **Update**, from the toolbar above the folders pane, not the details pane. See

[Updating the Workspace](#) on page 78.

### Go to root

Have the details pane display the contents of the depot’s (and workspace’s) top-level directory.

## Up one level

Have the details pane display the contents of the current directory's parent.

## Edit

Open a text editor on the currently selected file. (Select exactly one file before invoking this command.) You can use environment variable `AC_EDITOR_GUI` or `EDITOR` to control which text editor gets invoked.

The File Browser automatically updates the file's status indicators after an edit session. If you have changed the file's contents, AccuRev automatically places a **(modified)** status indicator in the **Status** column.

Note: you can also change the contents of files using operating system commands, scripts, and third-party tools, including integrated development environments (IDEs). In this case, invoke the command **View > Refresh** or press function key **F5** to update the file's status indicators.

## Add to Depot

Place the selected files/directories under version control. All the selected objects must currently have **(external)** status. For each one, AccuRev creates version 1 in the workspace stream.

## Keep

Create a new version in the workspace stream for each selected file. All of the selected objects must be file elements; none can have **(external)** status and none can be a directory.

Typically, you invoke the **Keep** command on files that you've been working on, and thus have **(modified)** status. But this is not a requirement. If you keep a file that you have not modified, a new version is created with identical contents.

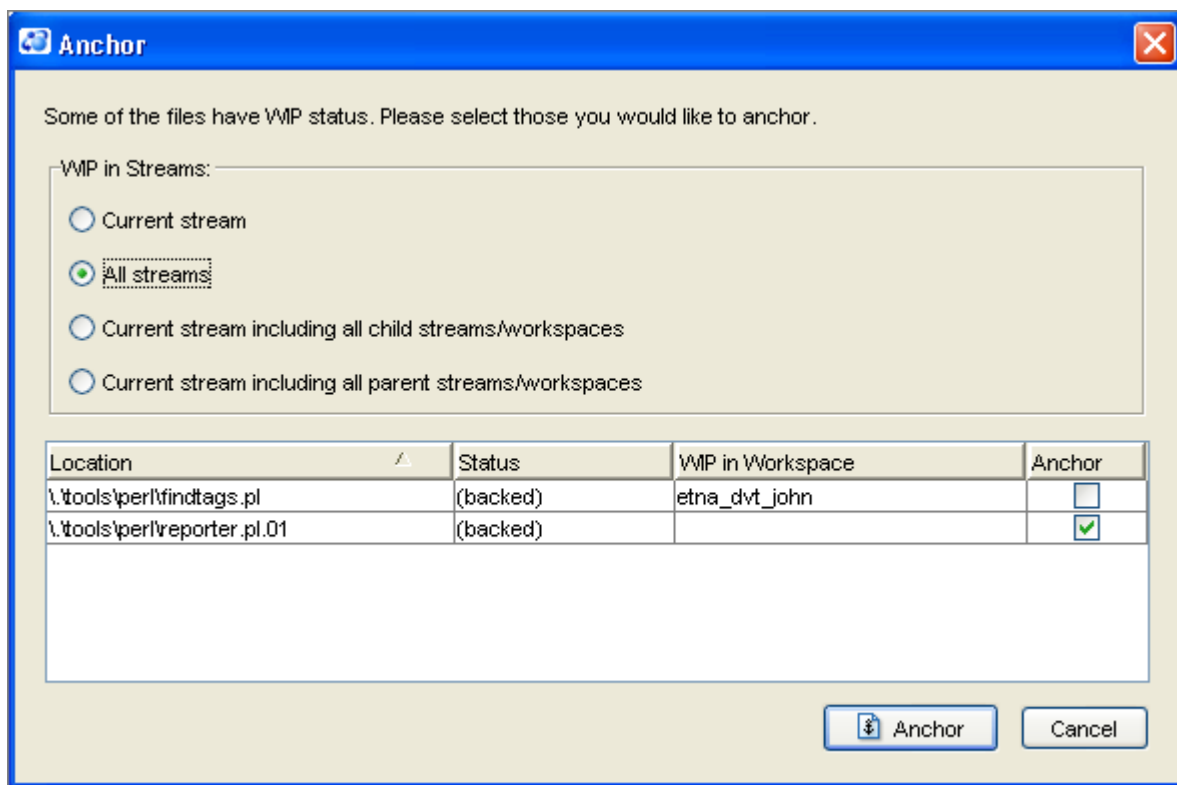
The first time you use the **Keep** command on a file, the file becomes active in the workspace ("is placed in the workspace's default group"). After that, you can modify and **Keep** a file as many times as you like. The file remains active in the workspace until you invoke the **Promote** or **Revert to Backed** command on it.

## Anchor

Make the selected files/directories active in the workspace ("place it in the workspace's default group"), without modifying them. Typically, you anchor a file in your workspace to prevent it from being overwritten with a newer version by a subsequent **Update** command. (**Update** overwrites inactive files only, not active ones.)

File(s) that you **Anchor** must currently be inactive in the workspace, from AccuRev's perspective; that is, they must have **(backed)** status. **Anchor** creates a new version in the workspace stream, identical in contents to the existing version. This new version simply records the fact that the file is officially active in the workspace.

AccuRev features anchor-required workspaces, in which you *must* **Anchor** a file before modifying it. When you anchor a set of elements in an anchor-required workspace, a dialog box appears if one or more of them are active in a sibling workspace:



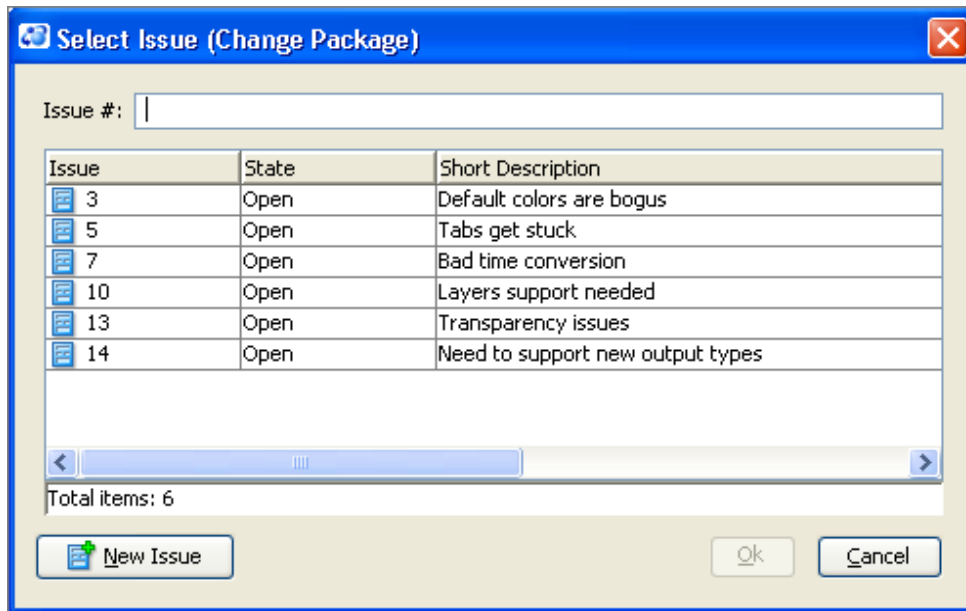
This enables you to select/deselect individual elements to be anchored. Note that the elements that are active in the sibling workspace are initially deselected. This makes it easy to avoid the situation where multiple users are working on the same file(s) concurrently.

## Promote

Send the active version of the selected files/directories to the workspace's backing stream (parent stream). The new version in the backing stream acts as a reference to the version created in your workspace. (In AccuRev parlance, it's a virtual version that "is an alias for" the version in your workspace.)

The selected element(s) become inactive in the workspace (are removed from the workspace's default group). The status of the elements becomes (**backed**).

**Integration between configuration management and issue management:** If either or both of the integrations between AccuRev's configuration management and issue management facilities is enabled, a particular AccuWork query is executed and its results displayed.



You can select one or more of the issue records and click **Ok**; or you can type one or more numbers (SPACE-separated) in the **Issue #** input field. The integration(s) record information about the **Promote** transaction in the issue record(s) you’ve designated.

For more information, see *Integrations Between Configuration Management and Issue Management* on page 220 of the *AccuWork Issue Management Manual*.

## Merge

For each selected file, perform a merge operation, involving two versions: the one in your workspace and the one in the workspace’s backing stream (parent stream). The resulting “merged version” is saved as a new version in the workspace stream.

The merge operation takes into account both the contents of the versions to be merged and the names of those versions. Thus, at the completion of a **Merge** command, a file might have a different name!

## Revert to Backed

For each selected file/directory, remove all changes you’ve made since the last time you **Promoted** it, or since the last **Update** of the workspace — whichever is more recent. For files, this includes both content changes and namespace changes. For directories, this includes namespace changes only.

The selected elements become inactive in the workspace (are removed from the workspace’s default group). The status of the elements becomes (**backed**).

Note: when you invoke this command in a dynamic stream, the depot’s **pre-promote-trig** trigger fires. That’s because, like a **Promote**, the command changes which version of the element appears in the stream.

## **Revert to Most Recent Version**

(on context menu, but not toolbar) For each file element with **(modified)** status, replace the file with the most recent version you created with **Keep**. Use this command when you've saved one or more intermediate versions of the file(s), and you want to discard further changes you've made since a **Keep**.

The selected elements remain active in the workspace. That is, they are *not* removed from the workspace's default group. The **(modified)** status of the elements changes to **(kept)**.

Note: if you've modified a file but not yet performed a **Keep** on it, this command works like **Revert to Backed**.

## **Show History**

Open a History Browser tab, containing the transactions involving the selected file or directory. See *The History Browser* on page 135.

## **Browse Versions**

Open a Version Browser tab, showing all the versions of the selected file or directory, and their interrelationships (ancestry). See *The Version Browser: Ancestry Tracking* on page 143.

## **Browse Stream Versions**

Open a Stream Version Browser tab for the selected element. This tab displays the depot's stream hierarchy, much like the StreamBrowser. But on this tab, each stream represents the *version* of the specified element that appears in that stream. See *The Stream Version Browser* on page 151.

## **Cut**

Mark the currently selected element to be moved to another directory in the same depot. To finish the relocation, right-click the destination directory (in either the folders or details pane) and select **Paste**.

Note: this command is available on context menus in both the details and folders panes.

## **Paste**

Specify the destination for an element that has been marked for relocation with the **Cut** command.

Note: this command is available on context menus in both the details and folders panes.

## **Rename**

Change the name of the selected file or directory. The new name can be in a different directory within the depot. Thus, this command can perform a “move” as well as a “rename”. This activates the object (includes it in the workspace's default group). Its status changes to **(kept)(member)**.

Note: if you change the contents of a file, then rename it without first performing a **Keep**, the version created by the **Rename** command does not contain the content changes, just the name change. The file's new status reflects this: **(modified)(member)**.

## Delete

Remove the selected files/directories from the workspace tree — the workspace's local disk storage. This command does not affect the depot — AccuRev still thinks the deleted objects should be there, so it continues to list them, with **(missing)** status.

To remove a file or directory from the depot's stream hierarchy, so that its removal will be reflected in other users' workspaces (after you **Promote** the change and they perform an **Update**), use the **Defunct** command instead of **Delete**.

## Defunct

Remove the selected files/directories from the workspace tree (local disk storage), and also change their status to **(defunct)**. This also activates the elements in the workspace stream — they're "in the act of being deleted". Accordingly, the elements also get **(kept)(member)** status.

When you **Promote** a defunct element, it disappears entirely from the workspace stream, and from the File Browser display. The element becomes **(defunct)**, and also active, in the backing stream.

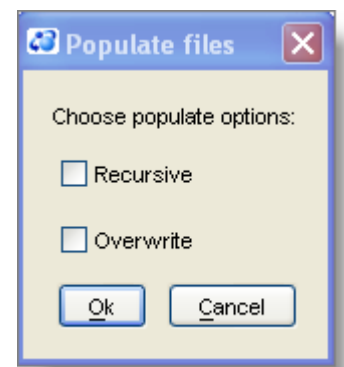
When you **Update** a workspace below a stream with a **(defunct)** file, the file is removed from the workspace's local disk storage. For a **(defunct)** directory, **Update** removes the entire subtree below that directory from the workspace's local disk storage.

## Populate

Copies the version that is currently in the workspace stream to the workspace tree. This has the effect of undoing an accidental deletion, or discarding edits that you have not yet preserved with **Keep**.

A dialog box controls these options:

- **Recursive** — For each selected element that is a directory, perform a Populate on that element and on all elements below it.
- **Overwrite** — For each file element, replace the file (if any) currently in the workspace tree. By default, Populate silently declines to overwrite an existing file.



## New File

Create an empty file in the current directory, and optionally place it under version control (**Add to Depot** checkbox).



## New Folder

Create an empty subdirectory in the current directory, and optionally place it under version control (**Add to Depot** checkbox).

## Send to Issue

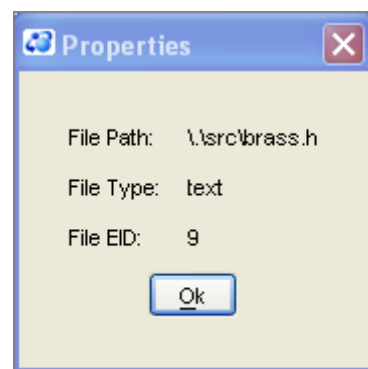
Record the selected version(s) in the change package section (Changes tab) of one or more issue records. The default query of the issues database is executed, and you are prompted to choose one or more of the records selected by the query. You can also create a new issue record, to which the selected version(s) will be sent.

## Send to Change Palette

(dynamic stream only): Load the selected versions into the Change Palette, so that they can be promoted to another stream. See *Using the Change Palette* on page 155.

## Properties

Displays information about the selected element: pathname, file type of current version, and element-ID.



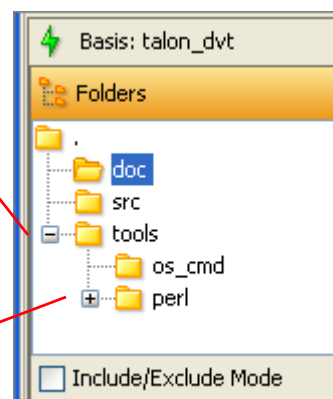
## Working in the Folders Pane

The folders pane includes a tree control, for navigating the folder (directory) hierarchy of a workspace. it works in the standard way:

- A “+” control indicates that the folder has a subhierarchy that is not currently displayed. Left-click the “+” control to open the subhierarchy.
- A “-” control indicates that the folder has a subhierarchy that is currently displayed (in part, or in its entirety). Left-click the “-” control to close the subhierarchy.

click on a - control to  
hide subdirectories

click on a + control to  
show subdirectories



Instead of left-clicking, you can double-click the folder icon or the folder name next to a “+” or “-” control.

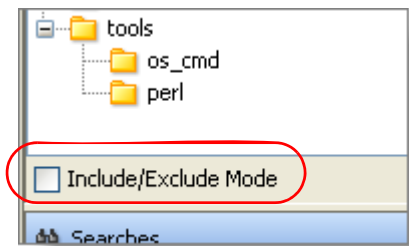
Note: on Unix/Linux systems, these controls are a right-arrowhead (instead of “+”), and a down-arrowhead (instead of “-”).

## Working in Include/Exclude Mode

By default, a workspace contains a copy of each file under version control in a particular depot. (More precisely, a workspace contains a copy of each version in a particular stream.) But sometimes, you don't want or need a copy of *every* file — for example, if your current assignment doesn't include “rebuilding the world” each night. The depot might contain many thousands of files, of which you might need only a small subset. The File Browser's include/exclude mode enables you to define and work with a subset of the depot's elements.

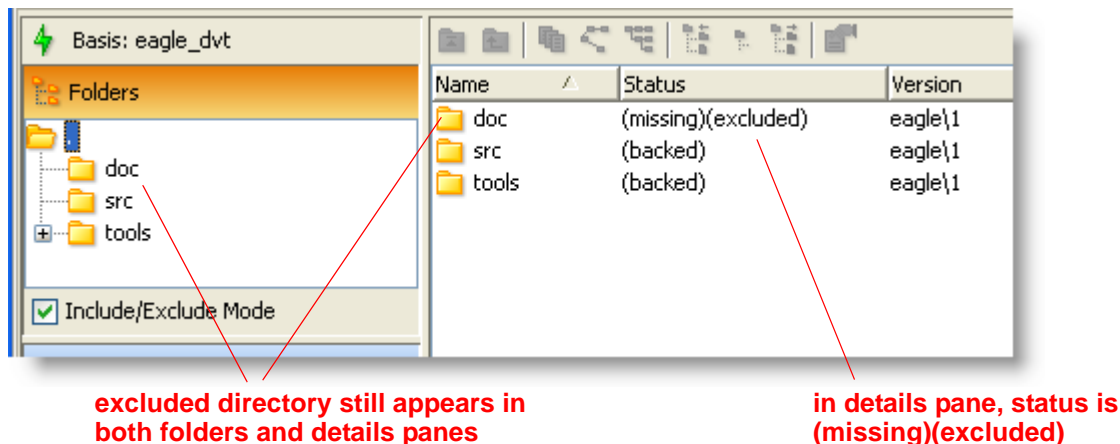
For more overview information on the include/exclude facility, see *Getting Only the Files You Need: the Include/Exclude Facility* on page 29 of the *AccuRev Concepts Manual*.

When you check (or clear) the **Include/Exclude Mode** checkbox, the File Browser immediately switches into — or out of — include/exclude mode. (There's no **Ok** button to press.) The File Browser works significantly differently in this mode, as described in the following sections.

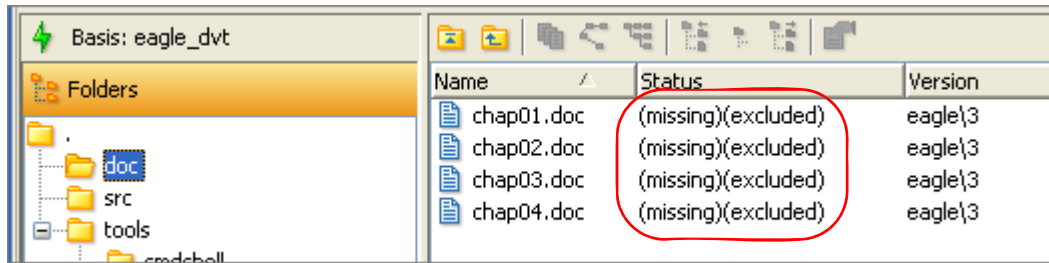


## How Include/Exclude Mode Changes the Folders and Details Panes

In include/exclude mode, both the folders and details panes show *all* elements in the workspace or stream. Elements that are currently excluded have **(missing)(excluded)** indicators in the Status column. For example, if top-level directory **doc** is excluded, the File Browser still shows it in both the folders and details panes:

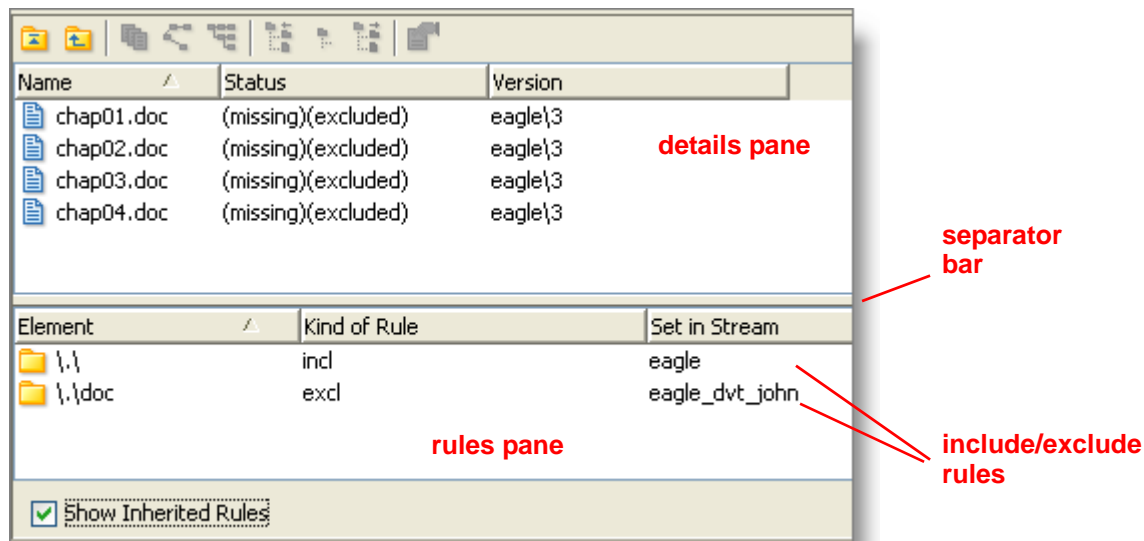


You can navigate into an excluded directory. All the elements therein will have **(missing)(excluded)** status.

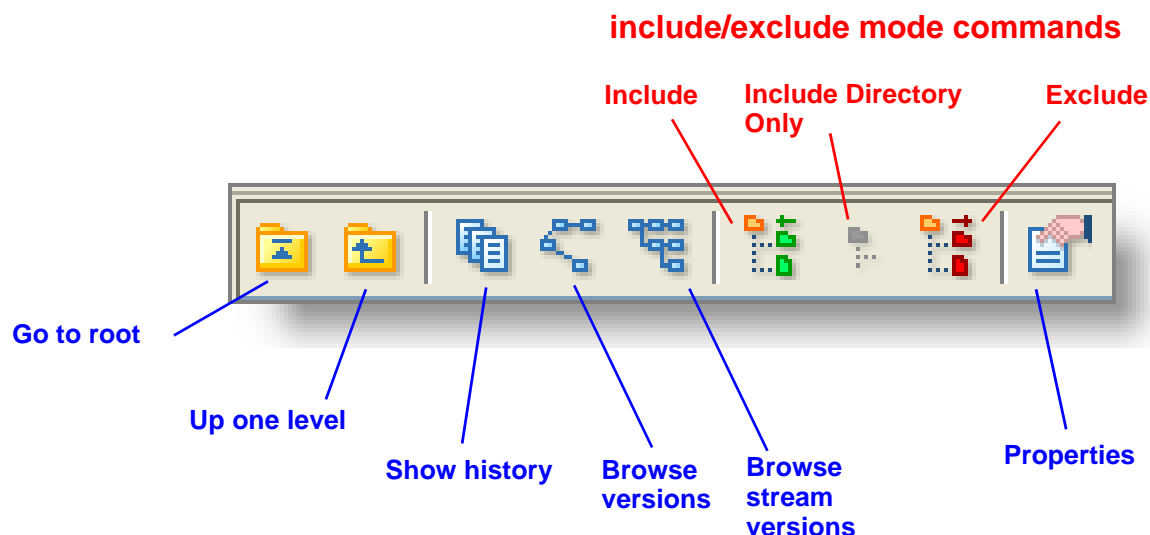


The File Browser gets a new pane, the rules pane, in include/exclude mode. This pane shows include and exclude rules that affect the current workspace or stream:

- **Show Inherited Rules** checkbox cleared: only the rules that were explicitly set for the current workspace
- **Show Inherited Rules** checkbox checked: also includes rules inherited from higher-level streams. The **Set in Stream** column indicates which higher-level stream.



The details pane has its own toolbar. When you switch to include/exclude mode, the toolbar changes: most of the buttons for AccuRev's version-control commands — **Keep**, **Promote**, **Merge**, etc.— disappear. Instead, buttons for the **Include**, **Include Directory Only**, and **Exclude** commands appear. A few of the standard toolbar buttons remain, to aid you in navigating the depot and determining element history.



### standard details pane commands

The sections below describe how to work in include/exclude mode.

## Adding Rules

In each depot, there is one hard-coded base rule: the depot's base stream has an include rule that


Element	Kind of Rule	Set in Stream
\.\	incl	eagle

specifies the depot's top-level (root) directory. This rule makes the depot's entire directory hierarchy visible in the depot's base stream.

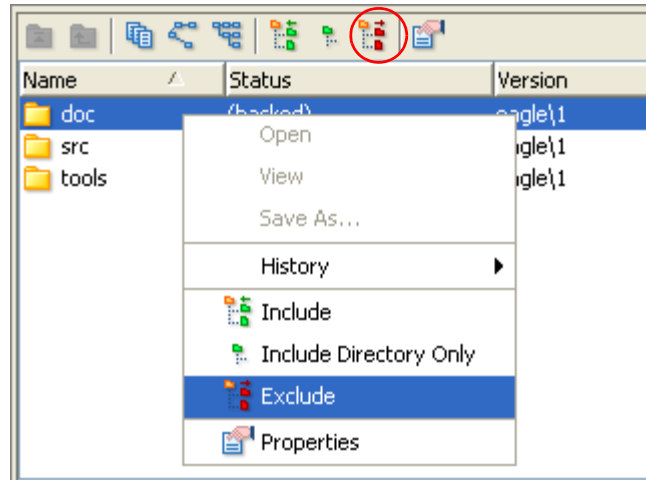
Any number of rules can be added. Each rule applies to a particular pathname within the depot's directory hierarchy ("Location"), and applies at a particular level in the depot's stream hierarchy ("Set in Stream"). A rule set in a dynamic stream gets inherited by lower-level streams; but a rule for the *same location* in a lower-level stream or workspace overrides a rule in a higher-level stream.

## Example: Excluding a Directory

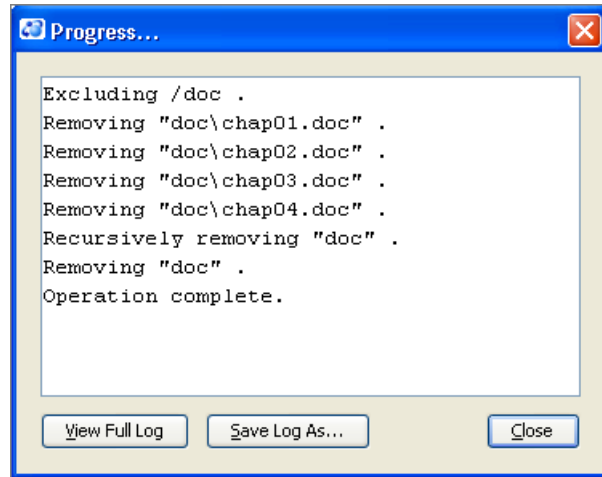
To exclude a particular directory from the current workspace or stream, add a new rule for it:

1. Make the directory appear in the details pane, by going to its parent directory in the folders pane.
2. Select the directory to be excluded and click the  **Exclude** button in the toolbar. Alternatively, right-click the directory and select **Exclude** from the context menu.

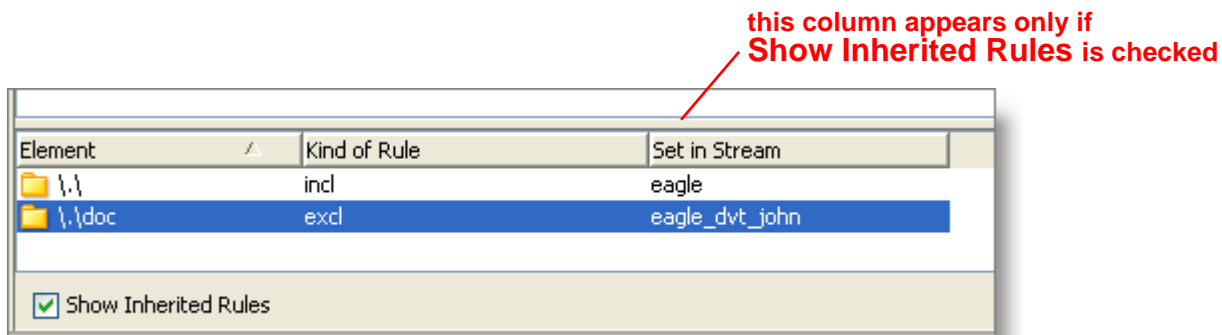
Alternative: if a rule already exists for the directory — set at the current level in the stream hierarchy or at a higher level — you can invoke **Exclude** from the context menu of that rule in the rules pane.



The entire directory tree below the specified directory is removed from the workspace or stream.



The new exclude rule now appears in the rules pane:



If you're in a dynamic stream, AccuRev reminds you that elements won't be removed from workspaces below that stream until they are **Update**'d. But keep in mind that element exclusion is instantly inherited by *streams* below that stream.

Note: if you want to remove some, but not all, of a directory tree from a workspace or stream, you must use an **Include Directory Only** rule, not an **Exclude** rule. The following examples discuss this further.

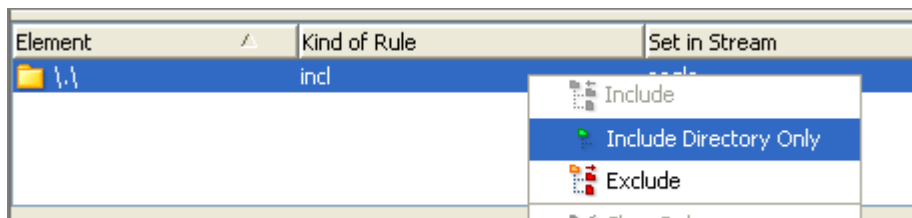
### Example 1: Simulating a Sparse Workspace

If you're accustomed to working in a "sparse" workspace (supported in older versions of AccuRev), you may want to do the include/exclude equivalent in a new workspace:

- remove all elements from the workspace
- add back to the workspace just the elements that you want to appear in it

The first step is accomplished by setting an **Include Directory Only** rule for the depot's top-level directory. Since the top-level directory never appears in the File Browser's details pane, you must invoke the command from the rules pane, not the details pane:

1. Make sure that **Show Inherited Rules** is checked in the rules pane, causing the depot's base rule to be displayed.
2. Right-click the base rule, and select **Include Directory Only**.






This creates a new rule, at the workspace level, for the top-level directory. This rule overrides, and replaces in the rules pane, the rule you just right-clicked:

Element	Kind of Rule	Set in Stream
\.	incldo	eagle_dvt_john

If you subsequently remove this new rule (see [Removing Rules](#) below), the overridden rule will reappear in the rules pane, and will reapply to your workspace.

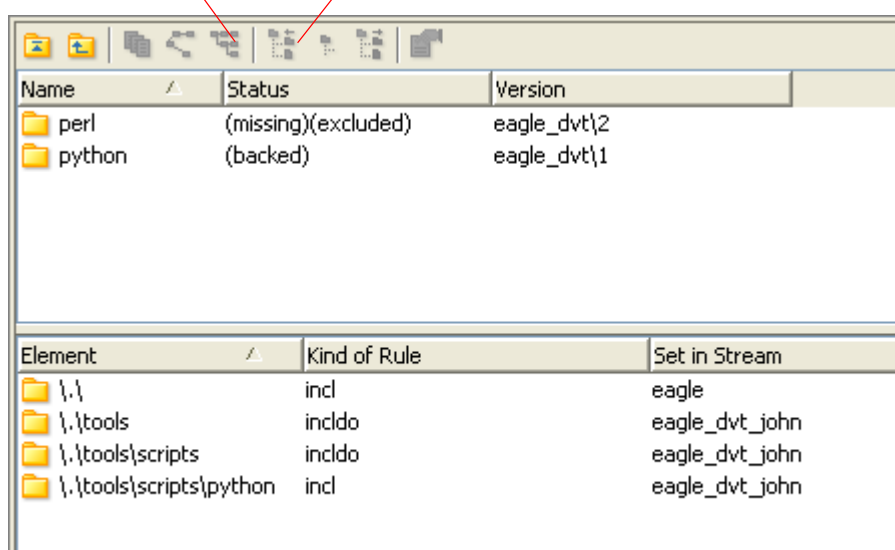
## Example 2: Excluding a Subtree

Sometimes, you want to exclude most, but not all, of a directory tree from a workspace or stream. For example, you might want to exclude the entire **tools** directory tree, except for subdirectory **python**. This is easy to accomplish, since the depot's entire directory hierarchy is always visible in include/exclude mode:

1. Add an  **Include Directory Only** rule for the **tools** directory.
2. Add an  **Include Directory Only** rule for the **scripts** directory.
3. Add an  **Include** rule for the **python** directory.

Here's the resulting details pane display:

**Include**      **Include Directory Only**



Name	Status	Version
perl	(missing)(excluded)	eagle_dvt\2
python	(backed)	eagle_dvt\1

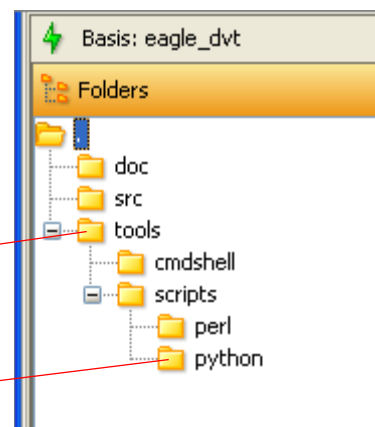
  

Element	Kind of Rule	Set in Stream
\\.	incl	eagle
\\.tools	incldo	eagle_dvt_john
\\.tools\scripts	incldo	eagle_dvt_john
\\.tools\scripts\python	incl	eagle_dvt_john

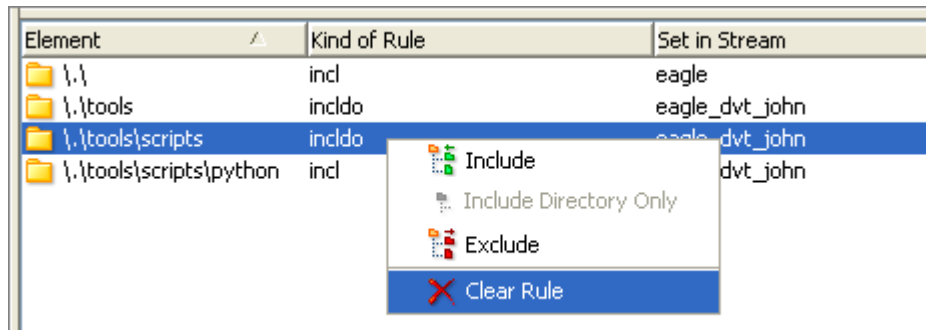
Note that the **Include Directory Only** command actually constitutes both an inclusion and an exclusion: “include the specified directory, but exclude everything under that directory”. Subsequent **Include Directory Only** commands can override, in part, the “exclude everything under” portion of the command.

## Removing Rules

To remove an existing rule that appears in the rules pane:



1. Make sure that the File Browser tab shows the stream or workspace in which the rule was explicitly set (**Set In Stream** column). If necessary, open a new File Browser on that stream or workspace.
2. Right-click the rule, and select **Clear Rule** from the context menu.



When you remove a rule from a stream, the effect is immediate on the stream itself and on streams below it. The effect does not take place on workspaces below the stream until they are **Update**'d.

When you remove a rule from a workspace, the effect is immediate on the workspace itself: files are copied into the workspace tree if you remove an exclude rule; files are deleted from the workspace tree if you remove an include rule.

## Leaving Include/Exclude Mode

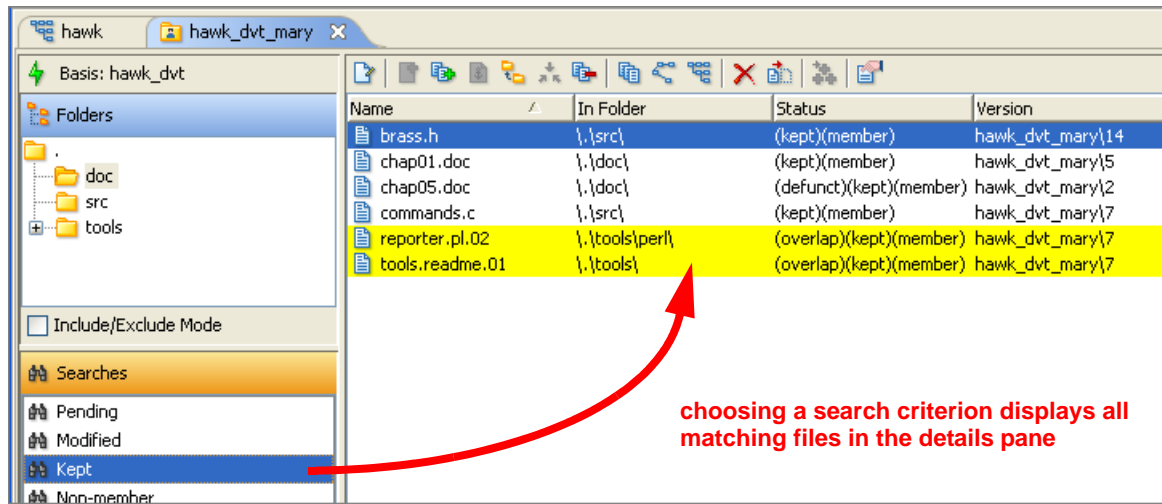
To leave include/exclude mode in a workspace or stream, clear the checkbox at the bottom of the folders pane. The rules pane disappears, and the details pane reverts to:

- *not* displaying elements that have been excluded from the current workspace or stream
- including its standard toolbar

## Working in the Searches Pane

The search criteria in the searches pane select elements according to their AccuRev status. For example, when you click **Kept** in the searches pane, AccuRev immediately launches a search for all elements — files and directories — in the workspace that are listed with the (**kept**) status indicator (and perhaps other indicators, too).





searches pane  
contains list of  
search criteria

choosing a search criterion displays all  
matching files in the details pane

The searches are not mutually exclusive: some objects may be selected by more than one search criterion — for example, **Kept** and **Pending**: every object selected by a **Kept** search will also appear in a **Pending** search.

The result of a search is a set of elements; this set is displayed in the details pane. You can perform commands on these elements, using the standard toolbars and menus.

## Search Criteria

The searches pane contains these search criteria:

### Pending

Selects elements whose status includes either **(modified)** or **(kept)**. These are elements that have had an AccuRev content or namespace change.

### Modified

Selects files whose status includes **(modified)**. This indicates a content change that has not yet been preserved with **Keep** (but may have followed a previous **Keep**).

### Kept

Selects files whose status includes **(kept)**. These are files whose content changes have all been preserved with **Keep**, or elements with namespace-level changes (**Rename** or **Defunct** command).

### Non-member

Selects files whose status includes **(modified)** but not **(member)**. These are files that have content changes, but are not in the workspace's default group. (They have not been activated with a **Keep** or **Anchor** command since the file's last update or promotion.)

## Default Group

Selects elements whose status includes (**member**). These are elements in the workspace's default group, for which you've entered one of these commands: **Keep**, **Rename**, **Defunct**, **Anchor**.

## Overlap

Selects the subset of Pending files whose status includes (**overlap**) — the current version in the parent (backing) stream is not an ancestor of the version in your workspace version. This means there might be content or namespace changes in the backing stream version that are not present in your workspace version. You need to perform a **Merge** with the backing stream version before you can **Promote** your workspace version.

## Deep Overlap

Reports (**overlap**) files in the current workspace or stream. Also reports (**overlap**) versions in the *parent* stream, in the *grandparent* stream, ... all the way up the depot hierarchy.

## Modified in Default Group

Selects elements whose status includes both (**modified**) and (**member**). This is the intersection of the sets of files selected by the Modified and Default Group search criteria.

## External

Selects files and directories that exist in the workspace tree (local file system), but have never been placed under version control with the **Add** command.

## Missing

Selects elements that *should* be present, but aren't. That is, there's a version of the file or directory in the workspace stream, but the file or directory was removed from the local file system (the workspace tree) by an operating system command or some non-AccuRev program.

In a sparse workspace, only a subset of the depot's elements are loaded into the workspace tree. In such a workspace, elements that have not been loaded have the status (**missing**), and so are listed by this search.

## Stranded

Selects the elements in the default group of a workspace or dynamic stream that have become stranded. A default group member becomes stranded when there is no pathname to the element in that workspace or stream.

Here's the most typical scenario for stranding a file in a workspace:

- Create a new version of a file with **Keep**. This places the file element in the workspace's default group.
- Remove the file's parent directory with **Defunct**. With the parent directory gone, there's no pathname in the workspace to the file element.

In the details pane, stranded files are identified by their element-IDs (since they have no pathname in the workspace or stream). For more information, see *Version Control of Namespace-Related Changes* on page 51 in *AccuRev Technical Notes*.

## Defunct

Selects the elements in the default group of a workspace or dynamic stream whose status is **(defunct)**.

## Controlling the Display of Element Names

Typically, the elements found by a search are located in a number of different folders, throughout the workspace. Accordingly, each element is identified by its complete depot-relative pathname.

You have a choice as to how to display the elements:

- Display the complete depot-relative pathname of the object as a single string (**Element** column).
- Display the object's simple name (**Name** column) separately from the pathname to its parent directory (**In Folder** column).

Element	Status	Version
\\tools\tools.readme	(kept)(member)	brass_dvt_john\11
\\tools\perl\reporter.pl	(kept)(member)	brass_dvt_john\10
\\doc\chap02.doc	(kept)(member)	brass_dvt_john\13
\\src\brass.h	(kept)(member)	brass_dvt_john\8

Name	In Folder	Status	Version
tools.readme	\\tools\	(kept)(member)	brass_dvt_john\11
reporter.pl	\\tools\perl\	(kept)(member)	brass_dvt_john\10
chap02.doc	\\doc\	(kept)(member)	brass_dvt_john\13
brass.h	\\src\	(kept)(member)	brass_dvt_john\8

Use the **Tools >**

**Preferences** command to make this choice (“Display of element name in tables”). It affects all instances of the File Browser, along with other tools that display element pathnames.

## Optimizations in Searches

Many of the File Browser's searches described above require that AccuRev consider every file in your workspace, even the ones that you haven't placed under version control (e.g. editor backup files, files produced by software builds). If your workspace contains many thousands of files, such operations can be time-consuming. AccuRev can use optimizations during a full-workspace search to significantly improve search performance. For details, see *Performance Optimizations* on page 81.

## Operating on Files Selected by a Search

You can work with the elements listed by a search using the same techniques and commands as described in *Working in the Details Pane* on page 50. That is, the details pane works the same way whether you've clicked in the folders pane or the searches pane. Here are a couple of notes:

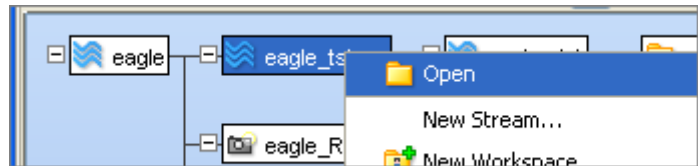
- Selecting all objects in the details pane can be particularly useful after a search. For example, to promote all files that you've saved with **Keep**: (1) click **Kept** in the searches pane, (2) click

one file in the details pane, then press **Ctrl-A**; this selects all the files, (3) invoke the **Promote** command.

- A file can disappear from the details pane if you change it in a search-related way. For example, after you promote all the files displayed by a **Kept** search, the files' status changes from **(kept)** to **(backed)**. Accordingly, the files disappear from the **Kept** search results.

## Displaying the Contents of a Dynamic Stream or Snapshot

In this chapter, we've assumed that you're using the File Browser to display the contents of a workspace. But the File Browser can also display the contents of a dynamic stream or a snapshot. In the StreamBrowser, right-click the dynamic stream or snapshot, and select **Open** from the context menu. (Alternatively, double-click the dynamic stream or snapshot.)



Dynamic streams and snapshots are different from workspaces. They exist only in the AccuRev repository, not on the disk storage of any client machine. And they don't contain actual files, but only database items that represent the element versions that have been created by the **Promote** command.

The versions that the File Browser displays as the contents of a dynamic stream or snapshot can have only these few statuses:

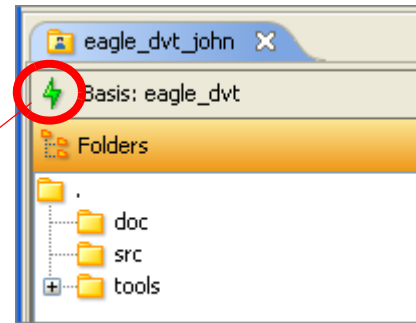
- **(member)** — the element is “active” in the stream (is in the stream's default group). This version has not yet been promoted to the stream's parent.
- **(overlap)** — the element is active in this stream, and the parent stream's version has changes that are not included in this stream's version. A merge is required before this stream's version can be promoted to the parent stream.
- **(defunct)** — the element is active in this stream: it was removed from a workspace with the **Defunct** command, and the change was then promoted to this stream.
- **(backed)** — the element is not active in this stream. This stream “inherits” the version of the element that appears in the parent stream.

The statuses that relate to an actual file in a user's workspace are not relevant in a dynamic stream or snapshot.

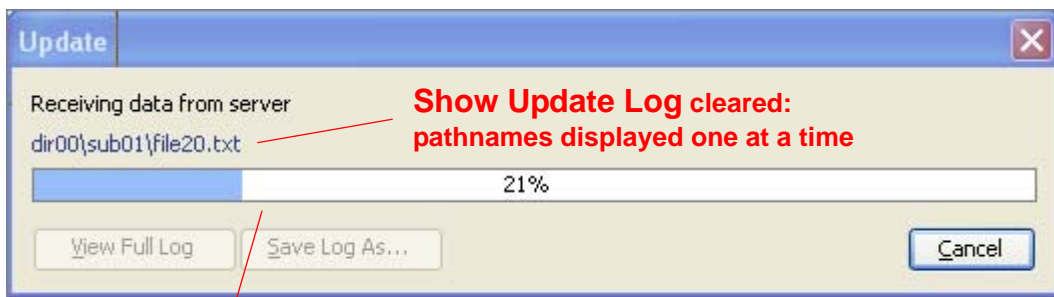
## Updating the Workspace

The principal purpose of an AccuRev workspace is to provide an *isolated* location for performing development tasks. The changes you make do not affect your colleagues until you decide to make them public, using the **Promote** command. Likewise, the changes that others make do not immediately affect your workspace. You must execute an **Update** command to bring versions created (and then promoted) by your colleagues into your workspace.

**Update  
command**



You invoke the **Update** command from the folders pane's toolbar. A progress window appears, displaying the pathnames of updated files — either one at a time or in a scrollable text field with a 2000-line capacity. The window also includes a progress bar, which eventually disappears to indicate that the **Update** has completed.



**progress bar  
disappears  
when Update is  
complete**



Use the **Show Update Log** preference (**Tools > Preferences**) to choose which way pathnames are displayed. Using the scrollable text field requires more system resources on the client machine, and can affect overall **Update** performance. Either way, when **Update** has finished, the **View Full Log** button provides access to a complete update log.

The following sections describe the workspace update process in detail.

## Kinds of Changes Involved in an Update

At its simplest, an update just copies versions of some files from your workspace's backing stream (its parent in the stream hierarchy) into the workspace. For example, your colleagues may have edited the contents of files **colors.java** and **main\_menu.java**, created new versions of them in their workspaces, then promoted the versions to the common backing stream. When you invoke **Update**, the new versions of those two files are copied from the depot to your workspace, overwriting the older versions of the file.

In addition to incorporating such content changes into your workspace, **Update** incorporates namespace changes:

- renaming of a file
- moving of a file to another directory
- creation of new files and directories
- defuncting of existing files and directories

AccuRev tracks namespace changes in the same way as content changes — by saving each change as a new version. If you **Rename** file **colors.java** to **colours.java**, the change is recorded as a new version of the file. Changing the name again, to **hues.java**, creates another new version.

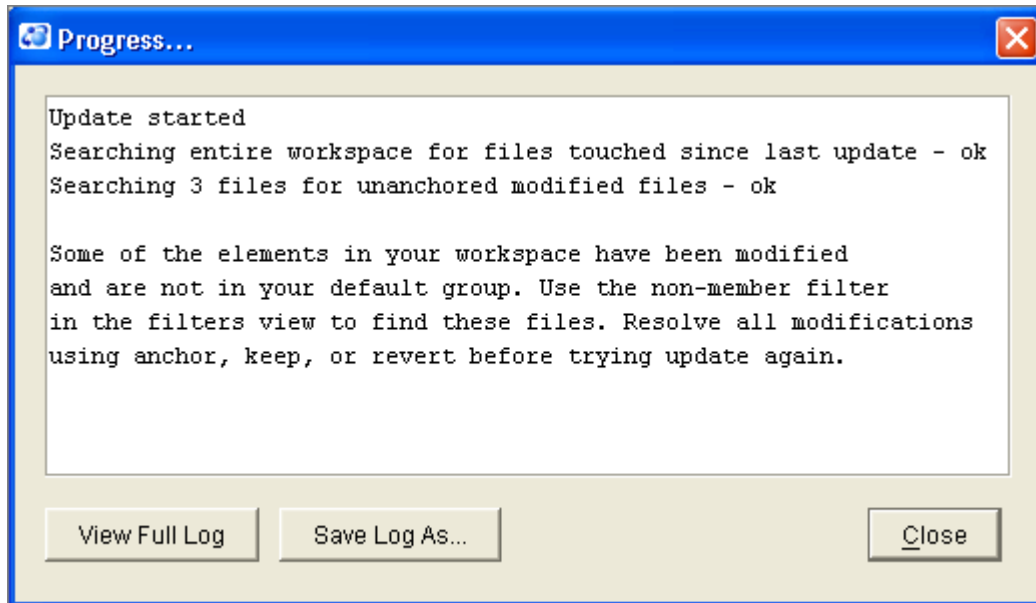
## Deciding Which Files to Update

Roughly speaking, **Update** partitions the entire collection of files in your workspace into two categories: (1) files that you're currently working on, and (2) all the others. The basic strategy is for **Update** to leave the files you're working on undisturbed, and to copy any new versions of the other files into the workspace. This enables the workspace to provide the safety of isolation, while still “keeping in touch” with other users' progress.

AccuRev uses the concept of default group to keep track of the files you're working on. (We often use the more informal term “active files” or “active elements” to describe the members of the default group.) Files are placed in the default group when you process them with such commands as **Keep** and **Rename**. The **Update** command won't bring a new version of a file (or directory) element into your workspace if that element is in the workspace's default group.

*Modified Files.* You might think that all elements *not* in the workspace's default group would be candidates for updating. Well, almost ... one of AccuRev's most useful features introduces a complexity here. Other version-control systems force you to perform a “check out” operation on a file before editing it; with AccuRev, you can edit any file at any time. But this means that you might be actively working on a file that is not yet a member of the workspace's default group — because you haven't yet issued a **Keep** on that file. Such files have the status “modified, but not in default group”. It would be wrong for **Update** to overwrite such files — you don't want this command to clobber changes you just made to a file!

**Update** handles this situation by refusing to proceed if the workspace contains any files whose status is “modified, but not in default group”.



Note: you *can* use AccuRev in a way that resembles those other version-control systems — see [Serial Development through Exclusive File Locking](#) on page 27 of the *AccuRev Concepts Manual*. A side-effect of this feature is that files don't get into the “modified, but not in default group” state.

**External Files.** A second factor that makes “candidate for updating” not exactly equivalent to “not in the default group”. A workspace can contain external files (and directories), which are not under version control. This typically includes text-editor backup files, results of software builds, mail messages, etc. External files won't ever be overwritten by an **Update**, since there are no versions of these files in the depot to be brought into the workspace. But there's a performance issue here: **Update** can spend a considerable amount of time analyzing the workspace's external files, before deciding that it doesn't need to worry about them.

The possible existence of modified files and/or external files in the workspace means that **Update** must examine the entire contents of the workspace before proceeding to update it. This is exactly the same as the situation encountered by the File Browser in Searches View. Accordingly, **Update** performance is enhanced through use of the same optimizations as search performance:

- A file is ignored if its pathname matches any of the patterns specified by environment variable `ACCUREV_IGNORE_ELEMS`.
- A file is ignored if its timestamp predates the time the workspace was last updated or otherwise searched for modified files (the workspace's scan threshold).

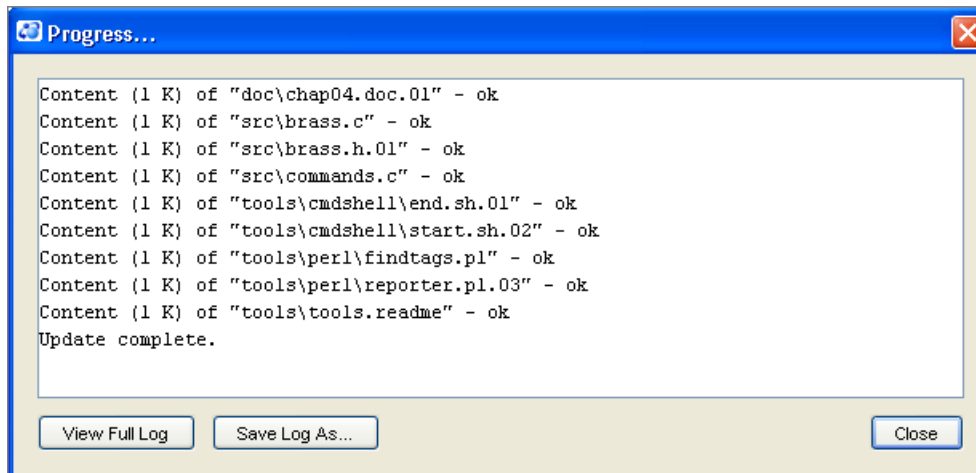
For more on these optimizations, see [Performance Optimizations](#) on page 81.

## Performing the Update

After determining the set of elements that are candidates for updating, **Update** determines the subset that actually *need* updating: the elements for which a more recent version exists in the backing stream. This step is efficient and speedy — **Update** need only consider the elements that

were involved in transactions recorded since the workspace's previous update. Only these transactions can contain changes that have not yet been incorporated into the workspace.

**Update** then replaces the workspace's versions with the newer backing-stream versions, logging its work in a separate window:



If there are many elements to be updated, the logging information scrolls out of view. You can click the **View Full Log** button to open a text editor on the entire log, or save it to a text file with the **Save Log As** button.

## Recording the Update

After it has completed the updating of versions in the workspace, **Update** updates two important workspace parameters:

- scan threshold (short for “modified files scan threshold”) — the time at which the workspace was last searched for modified files. The more recent this value, the more effective the timestamp optimization invoked both by **Update** and by certain File Browser searches. (See *Timestamp Optimization* on page 82.)
- update level (short for “update transaction level”) — the number of the depot’s most recent transaction. After an update, the workspace is “up to date as of transaction  $N$ ”;  $N$  is the workspace’s update level. The higher this value, the fewer transactions your next invocation of **Update** will need to examine, in order to determine which elements need to be updated.

(For a more detailed description of how **Update** does its work, see *The Update Algorithm* on page 45 of *AccuRev Technical Notes*.)

## Performance Optimizations

This section discusses two optional facilities that you can use to improve your AccuRev GUI experience. The timestamp optimization facility provides improved performance; the pathname optimization facility reduces “clutter” by hiding objects that you don’t care about, such as text-editor backup files.



Timestamp optimization is enabled through a checkbox, which affects both your work in the folders pane and in the searches pane. Pathname optimization is controlled through an environment variable, `ACCUREV_IGNORE_ELEMS`, which is read by the GUI when it starts.

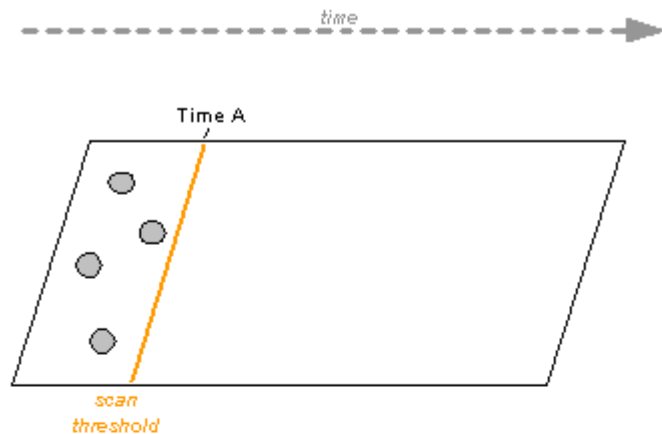
## Timestamp Optimization

In various situations, AccuRev searches your entire workspace tree for modified files:

- In a **Modified** search, the checksum of each file element in the workspace tree must be compared to the checksum of the version in the workspace stream.
- The **Pending**, **Non-member**, **Overlap**, and **Deep Overlap** searches also involve a search for modified files throughout the workspace tree.
- The **Update** command starts by performing a **Non-member** search, and refuses to proceed if it finds any modified files that are not recorded in the workspace's default group.

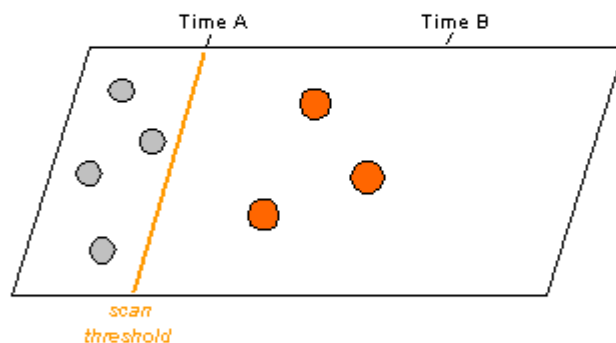
AccuRev keeps track of such modified-file searches, maintaining an ever-advancing scan threshold for each workspace. After a successful update, the scan threshold is advanced to the time that the **Update** command began. After any of the searches listed above, the scan threshold is advanced beyond the time of the most recent **Update**, to the point in time just before the timestamp on the oldest non-member modified file that the search located.

A successful Update at Time A advances the scan threshold to the current time. At this point, the workspace does not contain any non-member modified files.

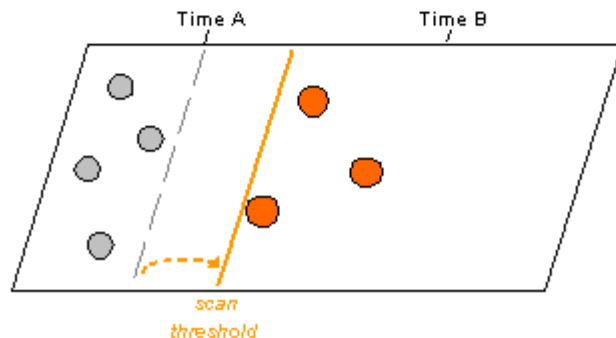


Between Time A and Time B, several files are modified, but not added to the workspace's default group.

At this time, an Update would fail, because of the non-member modified files.



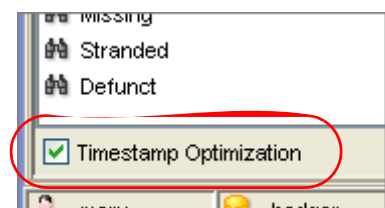
A search for non-member modified files advances the scan threshold to the point just before the oldest such file in the workspace.



The intended effect of these manipulations is to set the scan threshold to the *latest* time for which this principle holds true:

**Timestamp Optimization Principle:** For file elements that are not in the workspace's default group, the timestamp of a modified file is later than the workspace's scan threshold.

AccuRev's timestamp optimization takes advantage of this principle: if the checkbox at the bottom of the File Browser tab is checked, AccuRev ignores files with timestamps preceding the scan threshold when it performs an **Update**, or any of the other commands listed above.



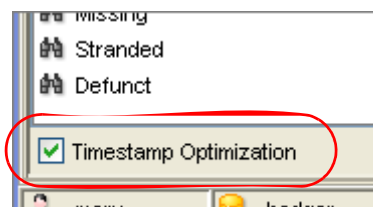
It should now be clear why the searches advance the scan threshold beyond the time of the last **Update**: making the threshold as late as possible enables AccuRev to ignore as many files as possible.

## Validity of the Optimization

The **timestamp optimization principle** is usually valid — but, unfortunately, not always. In general, if the only way you change version-controlled files is with text editors and build tools, the principle will be valid: each new change gets timestamped with the current time. But there are tools that can introduce “a new change with an old timestamp” into a workspace:

- The operating system’s “copy file” command can preserve old timestamps when creating a new copy of a file. Similarly, the **tar** (Unix) and **zip** (Unix and Windows) utilities can preserve old timestamps when they copy files out of an archive.
- If environment variable `ACCUREV_USE_MOD_TIME` is set, the AccuRev commands **co**, **pop**, **purge**, **revert**, and **update** preserve timestamps when copying versions from the repository into a workspace.
- Less likely but possible, a severely-lagging system clock on an AccuRev client machine can cause edited files to get timestamps that precede the most recent update. (AccuRev commands won’t execute if the client machine’s clock is not synchronized with the server machine’s clock. But something bad might happen to the client machine’s clock at a time when no AccuRev commands are being executed.)

If either of these situations applies to you, clear the **Timestamp Optimization** checkbox before performing an **Update** or any of the relevant searches. Turning off timestamp optimization can slow performance significantly, but it guarantees that no modified file will be overlooked because of a misleading timestamp.



If you know exactly which modified files have old timestamps, you don’t need to turn off the timestamp optimization. Instead, just update the timestamps to the current time, using the CLI command **accurev touch** (see page 184 in the *AccuRev User’s Manual (CLI)*).

## Pathname Optimization

You can configure the File Browser to ignore external files and directories, based on their pathnames. For example, you might know that files with the **.exe** suffix are not version-controlled, or that the entire contents of directories whose names start with **build\_** are not version-controlled. Ignored external files are omitted from the details pane, both when the File Browser displays the contents of a directory and when it displays the results of a search.

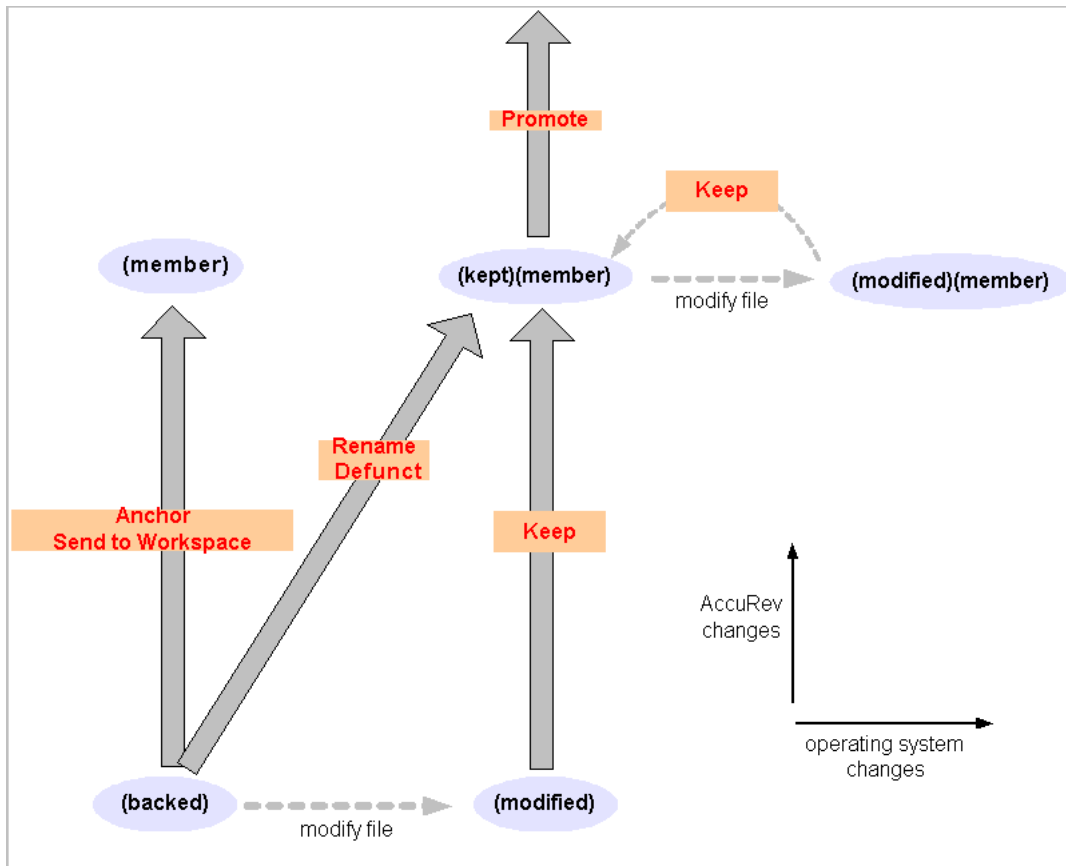
The environment variable `ACCUREV_IGNORE_ELEMS` controls this optimization. For example, you might set this variable to the following value before starting the AccuRev GUI:

```
*.exe */build_*
```

For details on this optimization, see *Using the `ACCUREV_IGNORE_ELEMS` Environment Variable* on page 27 in *AccuRev Technical Notes*.

## Appendix: File Statuses and Searches

The following diagram pulls most of the file statuses together, showing how an element's changing status indicates its development progress. This diagram leverages the central AccuRev concept of promotion, which suggests that as more work is performed on an element, it “rises to a higher level” of development. The diagram also depicts the fact that AccuRev tracks changes to an element in two “dimensions”: it records changes made by AccuRev commands as new versions in the depot; and it detects that a file's contents have changed at the operating system level.



This diagram captures the following facts:

- Before you start working on an element, it's unchanged along both the AccuRev dimension and the operating-system dimension. Its status is **(backed)**.
- When you modify the contents of a file, it changes along the operating-system dimension only, and becomes **(modified)**.
- The following involve changes along the AccuRev dimension. This makes the element active in the workspace, so its status indicates that it is a **(member)** of the default group.
  - When you **Keep** a file that you've modified, its status becomes **(kept)**.
  - An element can also achieve **(kept)** status through a namespace change — **Rename** or **Defunct** command. This is a change along both the AccuRev and operating-system

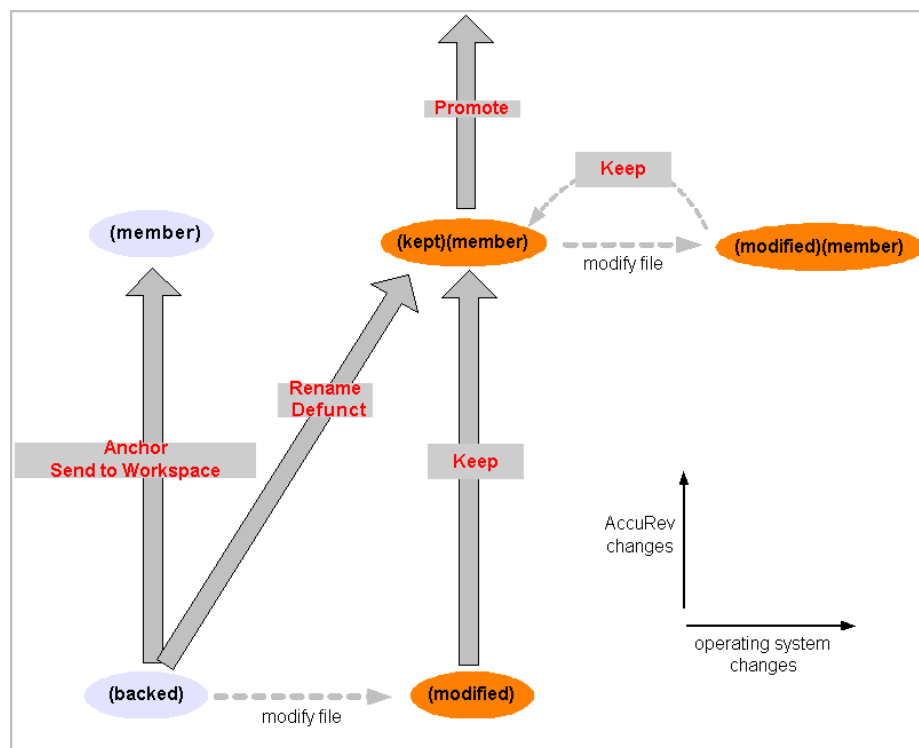
dimensions. (The CLI command **undefunct**, which reinstates a previously defuncted object, works similarly. This command is not in the GUI.)

- The **Anchor** command change a file along the AccuRev dimension only. The file's status becomes **(member)**.
- You can create any number of intermediate versions of a file in the workspace, by repeatedly modifying the file then **Keeping** it. Throughout this process, the file remains a **(member)** of the workspace's default group. In addition, its status toggles between **(kept)** and **(modified)**.
- You can **Promote** an element whose status is **(kept)**. This returns the element's status to **(backed)** — the workspace now uses the newly promoted version in the backing stream.

The diagram makes it easy (we hope!) to see how each search criterion works:

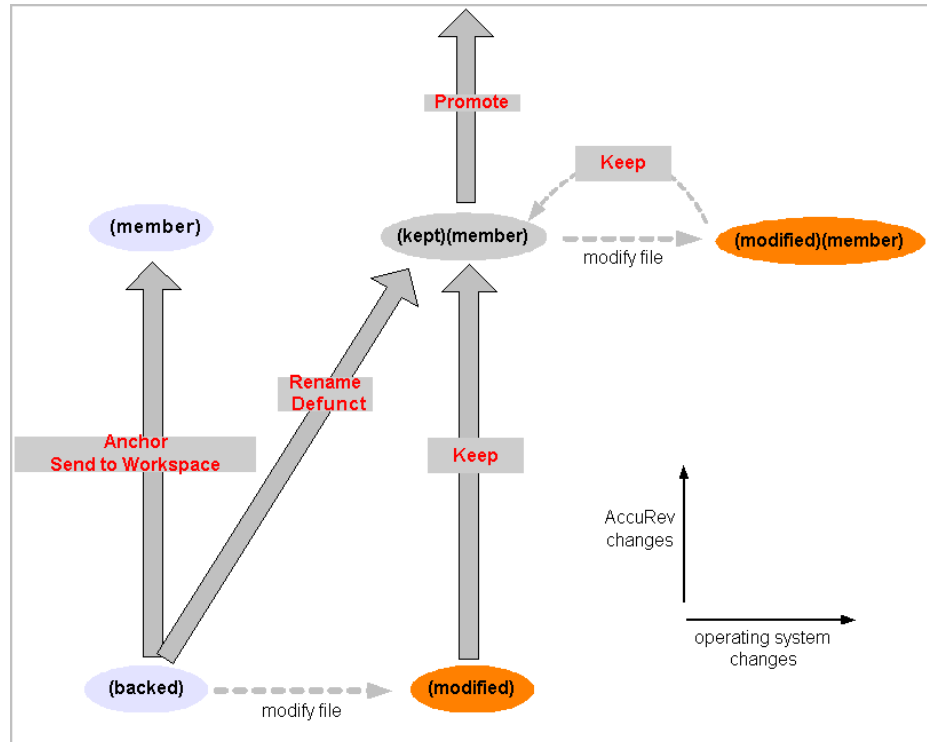
### Pending

Selects elements whose status includes either **(modified)** or **(kept)**. These are elements that have had an AccuRev content or namespace change, or a change in file contents.



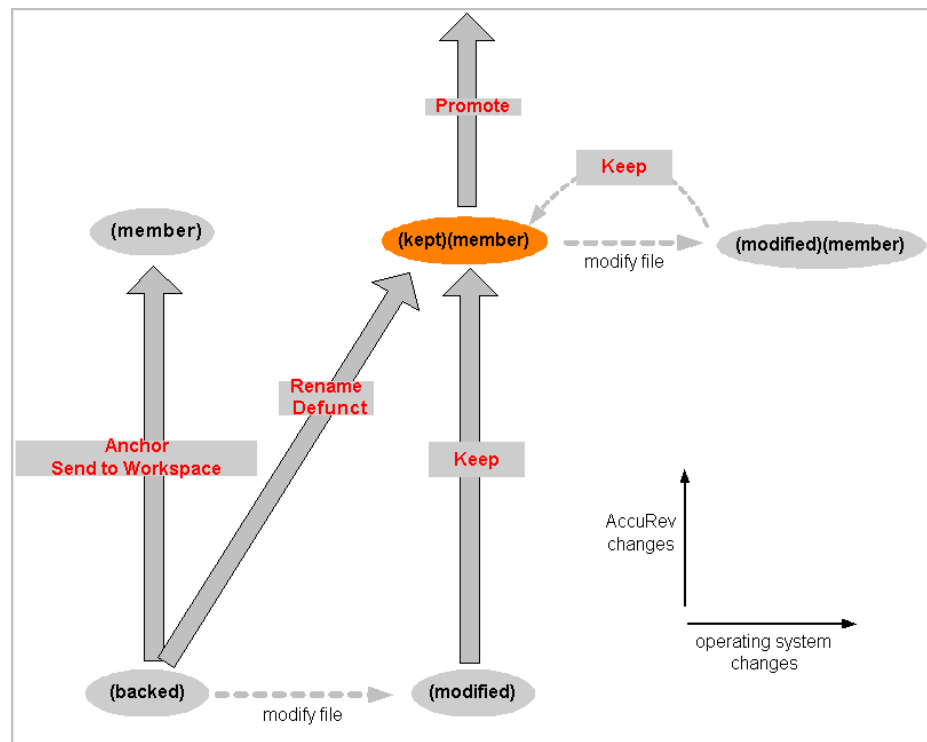
## Modified

Selects files whose status includes **(modified)**. This indicates a content change that has not yet been preserved with **Keep** (but may have followed a previous **Keep**).



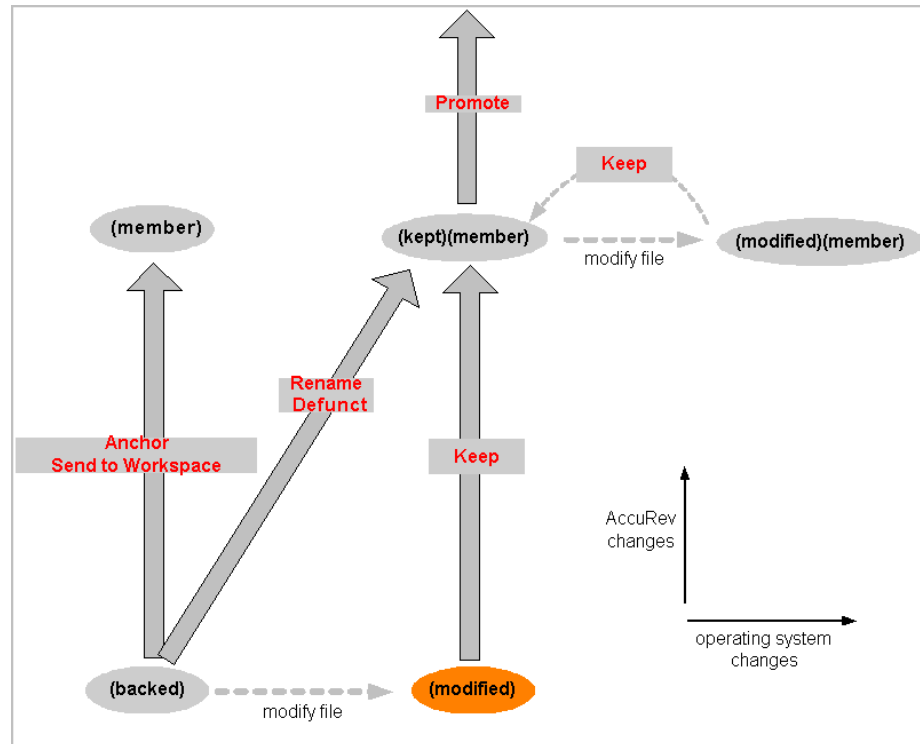
## Kept

Selects files whose status includes **(kept)**. These are files whose content changes have all been preserved with **Keep**, or elements with namespace-level changes (**Rename** or **Defunct** command).



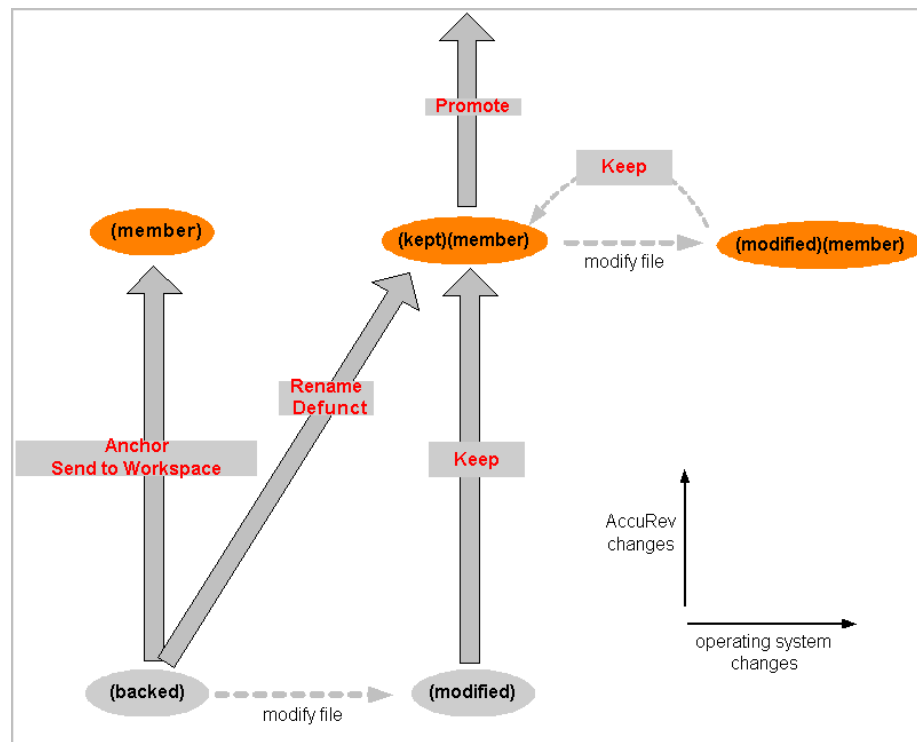
## Non-member

Selects files whose status includes **(modified)** but not **(member)**. These are files that have content changes, but are not in the workspace's default group. (They have not been activated with a **Keep** or **Anchor** command since the file's last update or promotion.)



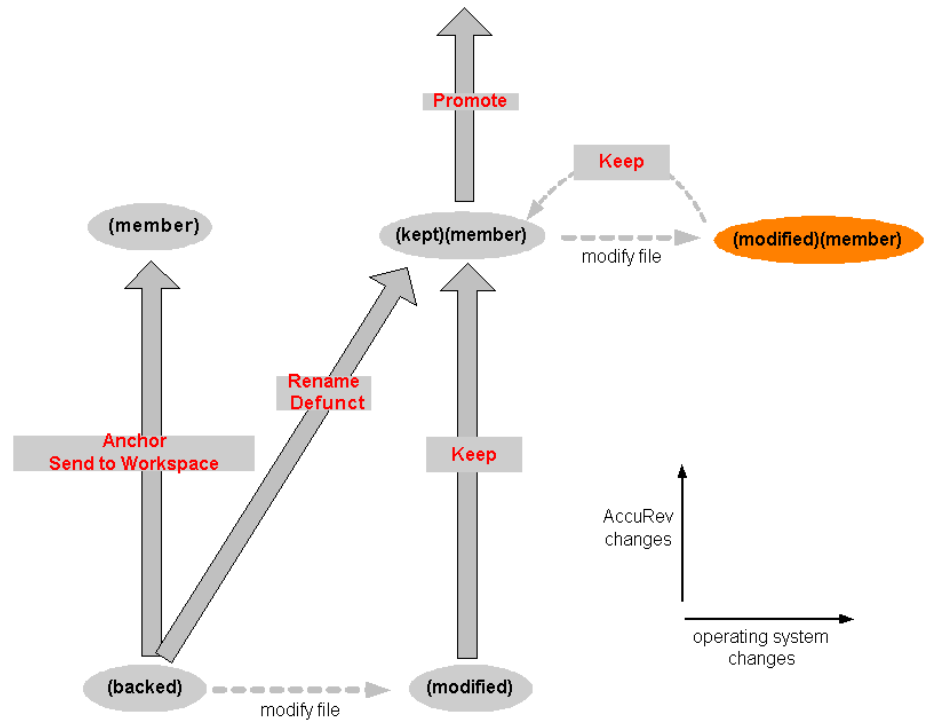
## Default Group

Selects elements whose status includes **(member)**. These are elements in the workspace's default group, for which you've entered one of these commands: **Keep**, **Rename**, **Defunct**, **Anchor**.



## Modified in Default Group

Selects elements whose status includes both **(modified)** and **(member)**. This is the intersection of the sets of files selected by the Modified and Default Group search criteria.



The status diagrams above do not account for these statuses: **Overlap**, **Deep Overlap**, **External**, **Missing**, **Stranded**, **Defunct**.






# The StreamBrowser

One of AccuRev's most powerful tools is the StreamBrowser®. It enables graphical control over the entire configuration management environment, in a way that is simple, flexible, and powerful. The StreamBrowser enables you to view and manipulate all the streams in a depot. This includes the streams that are built into workspaces; it also includes snapshots, which are "frozen streams".

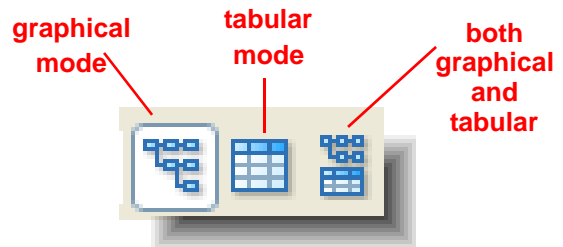
## Opening a StreamBrowser Tab

To open a StreamBrowser tab, select **View > Streams** from the command menu. You can also click the  **View Streams** toolbar button, or select **View > Streams** from the context menu of a depot on a Depots tab. The StreamBrowser tab displays the streams of the current depot (the one whose name is displayed at the bottom of the GUI window). If there is no current depot, AccuRev prompts you to select one.

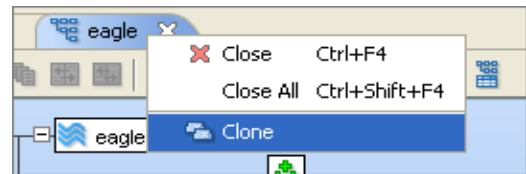


The StreamBrowser offers the following display modes, which you select using toolbar buttons:

- Graphical display of the depot's stream hierarchy.
- Tabular display of the depot's stream hierarchy.
- Both of the above: the tab is divided into a graphical pane (above) and a tabular pane (below). A movable separator bar enables you to adjust the allocation of screen space to the panes.

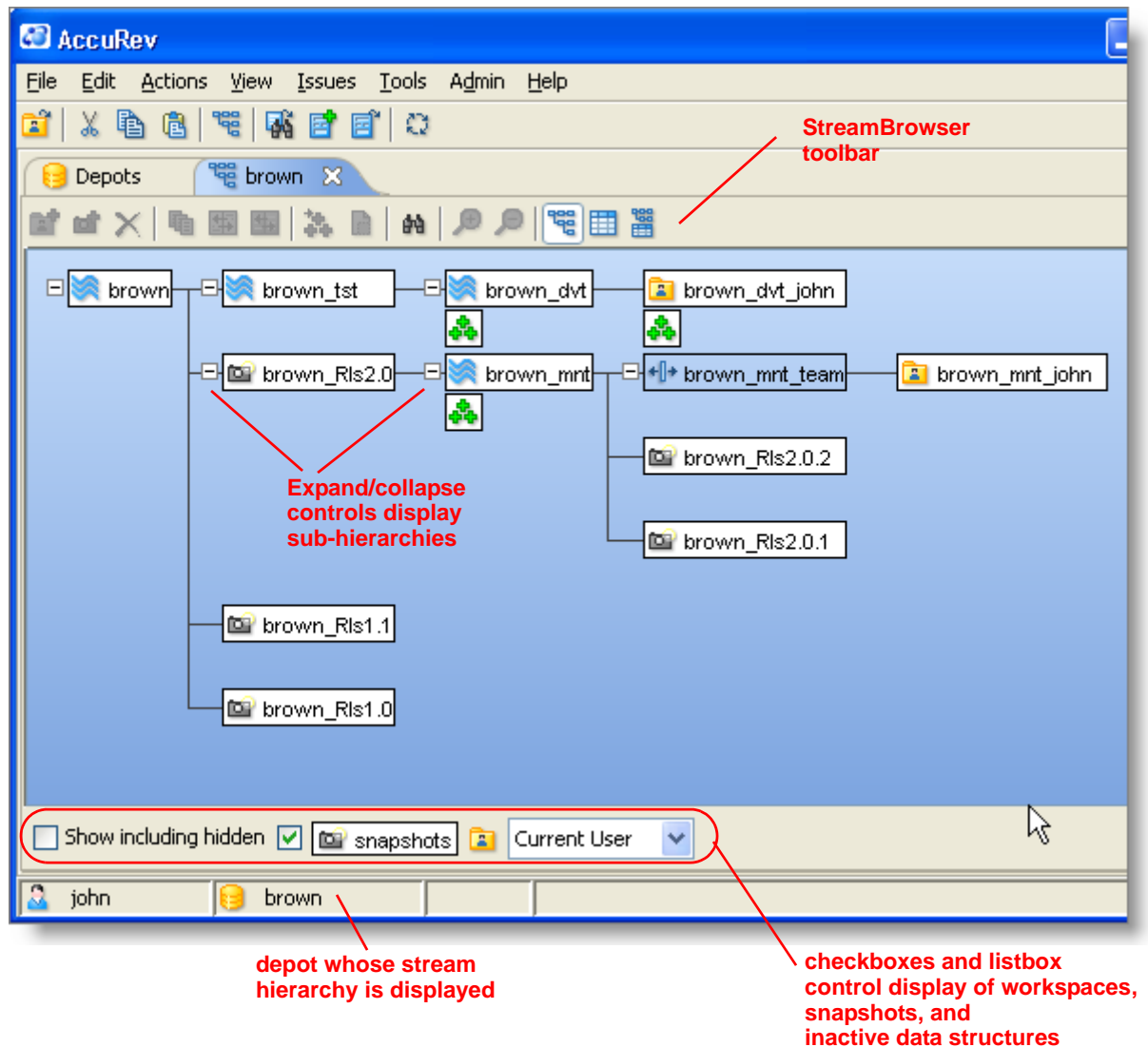



You can have any number of StreamBrowser tabs open at the same time, each for a different depot. You can also open multiple StreamBrowser tabs on the *same* depot, by invoking the **Clone** command on the context menu of the tab control.





## Graphical StreamBrowser Display

The graphical display of a depot's stream hierarchy is organized as follows:




- The depot's root stream (top-level) is at the left edge. The  **Zoom In** command enables you to restrict the display to any subhierarchy, for simplicity and improved performance.
- A given stream's children appear to its right; the children are arranged vertically, in this order:
  - Workspaces, in alphabetical order
  - Dynamic streams with no basis time, ordered by creation time (most recent first)
  - Dynamic Stream / Snapshot with basis times, ordered by basis time (most recent first)
- Each stream is annotated with an icon that indicates the kind of stream:

 dynamic ("normal") stream

 time-based dynamic stream

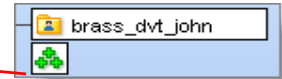
 workspace stream

 static stream (snapshot)

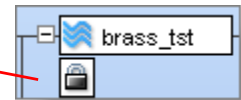
 passthrough stream

- Each stream or workspace that has active elements — the default group of the stream or workspace contains one or more elements — is displayed with a special control that opens and closes a window showing the default group. See *Displaying and Working with the Default Group* on page 98.
- Each stream can have a lock, which prevents certain operations involving that stream from being performed. If a lock exists on a stream, a special control appears, which provides access to the **Locks** dialog box.

open/close  
default  
group



open  
Locks  
dialog box



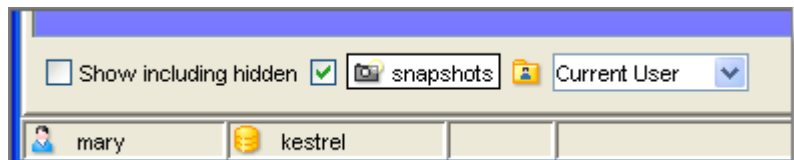
## Controlling the Display of Streams, Snapshots, and Workspaces

Initially, the StreamBrowser displays all of the depot's currently active dynamic streams, along with currently active snapshots. It doesn't display any workspace streams, nor any data structures that you have deactivated with the **Remove** command on its context menu.

Note: for simplicity, this chapter will usually refer to dynamic streams as “streams”, and to workspace streams as “workspaces”.

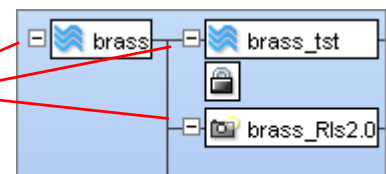
Using the checkboxes the listbox at the bottom of the StreamBrowser tab, you can:

- Reveal/hide the data structures that have been **Removed**. (The **Remove** command doesn't actually delete anything from the depot; the data structure just becomes invisible and inactive.)
- Control the display of the depot's snapshots (but perhaps not those that have been **Remove**'d).
- Control the display of the depot's workspaces (but perhaps not those that have been **Removed**). The list box contains the choices **Current User**, **All Workspaces**, **No Workspaces**, and the principal-name of each AccuRev user.

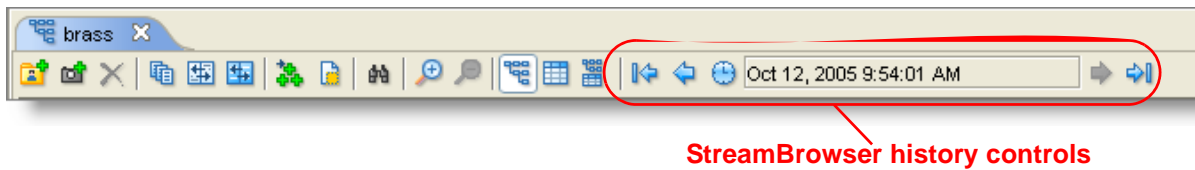


Each stream that has “children” (workspaces and/or snapshots) is displayed with an expand/collapse control. Collapsing causes the entire hierarchy below the stream to disappear from the screen. This affects the StreamBrowser display only. It does not affect the operation of the stream in any way.

expand/  
collapse  
control




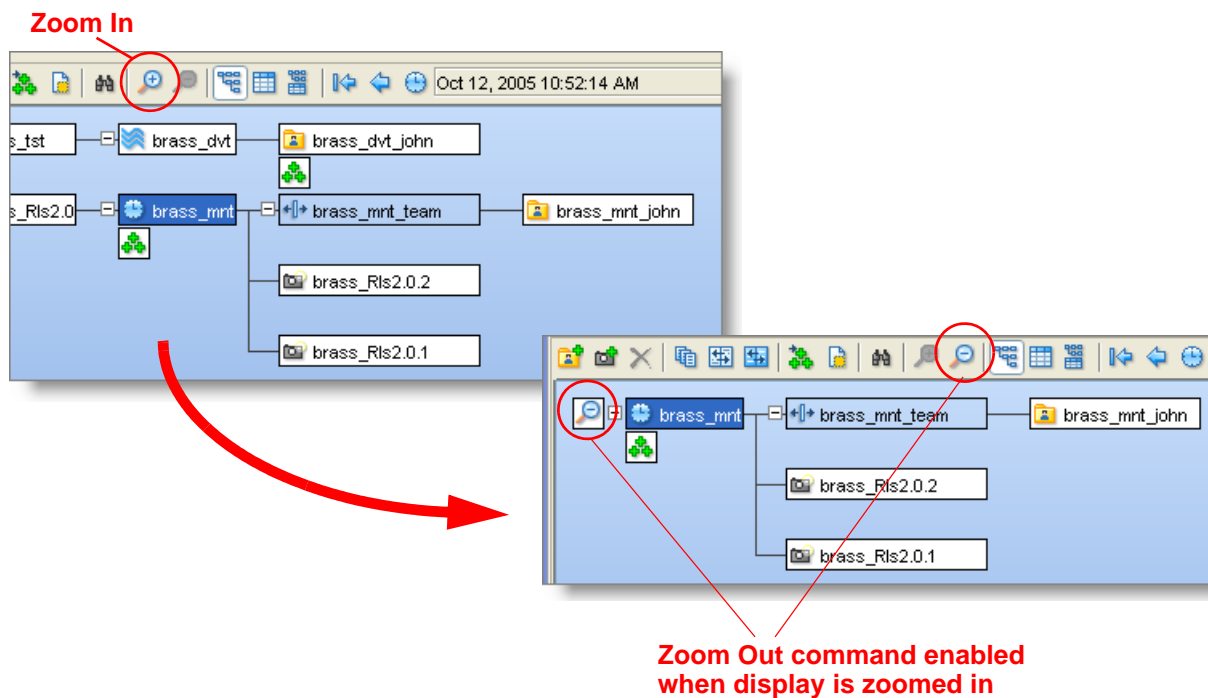
If the **Enable Stream Browser History** preference is checked (**Tools > Preferences** command), a set of history controls are added to the toolbar:



This enables you to view the stream hierarchy as it existed at any point in the past. See *Stream History* on page 112.


## Zooming In on a Subhierarchy

When you select any stream (except the depot's root stream), the  **Zoom In** command is enabled. Executing this command restricts the StreamBrowser display to the subhierarchy below the selected stream. (Think of the selected stream as becoming the “temporary root” of the hierarchy.)



Zooming in both simplifies the display and improves StreamBrowser performance. For example, refreshing the display involves determining the default-group status and the lock status for each displayed stream. When you have fewer streams displayed, this determination proceeds more quickly.

Zooming in does not affect AccuRev operations, only the StreamBrowser display. In particular, inheritance of versions and include/exclude rules is not affected. And zooming in does not prevent you from promoting versions from the “temporary root” stream to its parent stream.

When the display is zoomed in, the  **Zoom Out** command is enabled in the StreamBrowser toolbar. In addition, a **Zoom Out** button appears next to the “temporary root” stream. Invoking this command returns the StreamBrowser to displaying the depot’s entire stream hierarchy.

The zoom setting for a depot is persistent. You can close the StreamBrowser tab or exit the AccuRev GUI; the next time you open a StreamBrowser tab on that depot, it will be displayed at the same zoom setting.

## Ensuring Performance Improvements

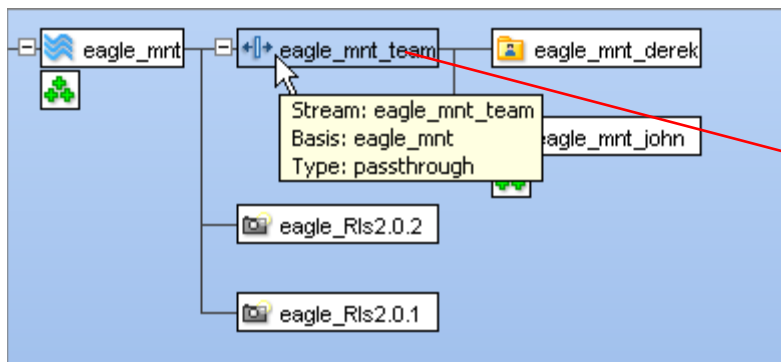
Zooming in on a particular subhierarchy provides better performance because the AccuRev GUI client software doesn’t need to retrieve (and repeatedly refresh) certain data from the AccuRev server regarding the streams outside that subhierarchy. But certain operations may cause the GUI to retrieve data for a larger subhierarchy, or for the entire stream hierarchy. For example, opening a File Browser on a workspace outside the zoomed-in subhierarchy causes the GUI to expand its focus and retrieve data for the larger (sub)hierarchy that includes that workspace. This may degrade StreamBrowser performance, even though it doesn’t actually display the streams in the expanded focus.

## Manipulating the Stream Hierarchy

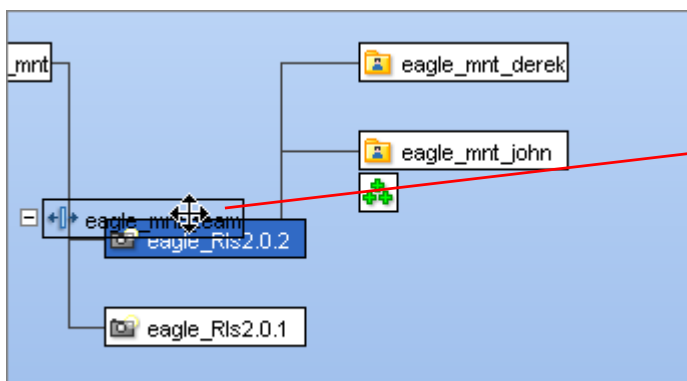
The StreamBrowser doesn’t just display a depot’s stream hierarchy — it’s also a tool for manipulating the hierarchy. You can add new streams, snapshots, and workspaces; you can rearrange the hierarchy, and you can remove (that is, hide) existing data structures.

## Rearranging Streams and Workspaces

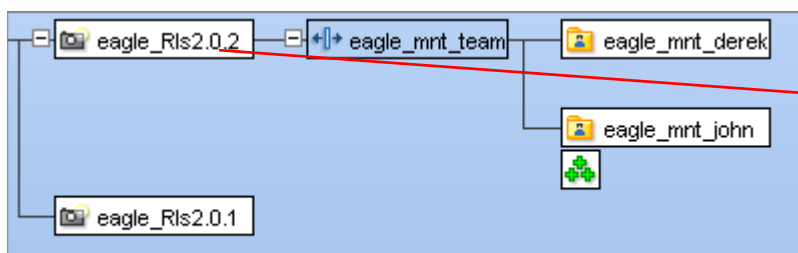
AccuRev lets you change the backing stream (parent stream) of any dynamic stream or workspace. The StreamBrowser makes it simple: you just drag-and-drop a stream or workspace from its current location in the hierarchy to its new parent. The entire subhierarchy moves to the new location:



1 click on a stream or workspace, and start dragging it



2 drop the data structure on its new "parent"



3 StreamBrowser moves entire subhierarchy to new location

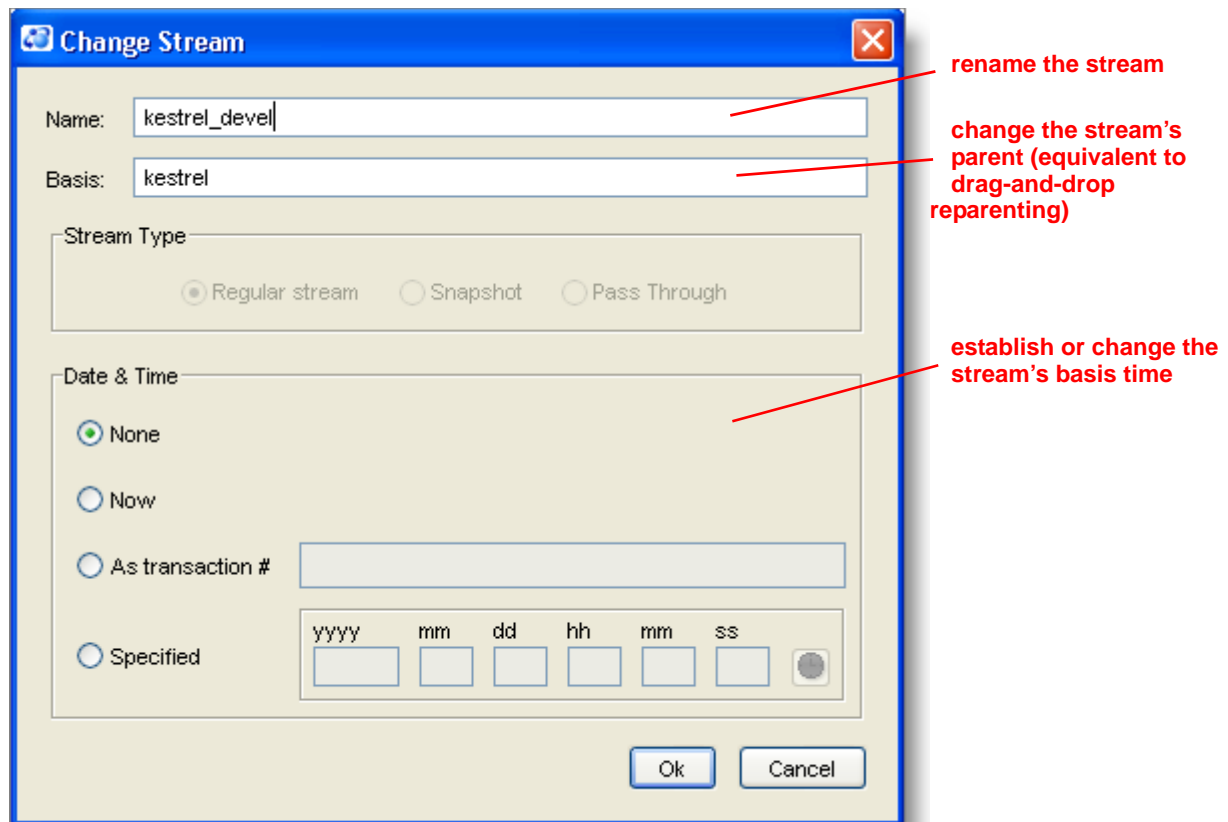
Changing a data structure's location in the stream hierarchy is called reparenting. You cannot reparent a snapshot; both the contents and the parentage of snapshot are fixed permanently.

Note: after you change the location of a workspace, be sure to **Update** it. This ensures that the workspace contains the correct set of versions, many of which it will inherit from its new parent. Likewise, after changing the location of a stream, all workspaces in the subhierarchy below that stream should be **Update**'d.

You cannot reparent (or change any of the other specifications) of a workspace created with the exclusive file locking feature. See [Creating a Workspace](#) on page 29.

## Reconfiguring a Stream

The drag-and-drop operation changes one property of a stream: its parent. You can also change a stream's properties using the **Change Stream** command on its context menu.



Although you can give a new name to an existing stream, you cannot proceed to create a new stream with the old name. The old name remains associated with its stream. The only way to reuse a stream name is to completely remove the stream's depot from the AccuRev repository, using the AccuRev administration utility, **maintain**. See *The 'maintain' Utility* on page 293 of the *AccuRev Administrator's Guide*.

By default, a stream inherits all versions from its parent, no matter when those versions were created. If you assign a basis time to a stream, it inherits only those versions created before the specified point in time. The most common use of a basis time is to create a permanent, unchanging snapshot of a particular stream.

Note: there's another way to restrict which versions get inherited from the parent stream: the include/exclude facility. You must use the File Browser to access this facility. See *Working in Include/Exclude Mode* on page 67.

## Creating New Streams, Snapshots, and Workspaces

The context menu of any data structure in the StreamBrowser includes commands for creating new structures at that point in the hierarchy. **New Stream** and **New Snapshot** display dialog boxes similar to that of **Change Stream**, discussed above. The **New Workspace** command invokes a wizard that steps you through the process of defining a new workspace: you specify a name and a location on disk; you can also make some optional settings, such as controlling how line endings in text files are to be handled.



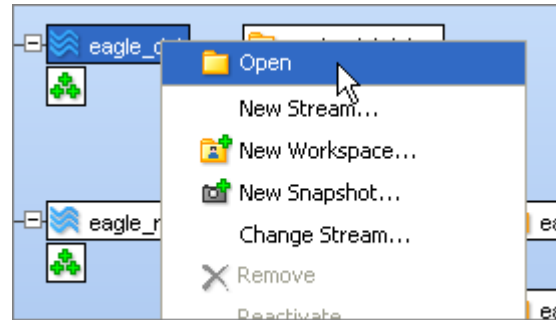
## Exploring the Contents of Streams

You can view the entire contents of any stream, snapshot, or workspace by invoking a File Browser. Double-click the data structure, or right-click it and select **Open** from the context menu.

For user workspaces, the File Browser shows the status of the files in the user's disk storage (the workspace tree). If a particular workspace's storage is not available to you, the data display may be incomplete.


For dynamic streams and snapshots, the display is always complete: the File Browser displays configuration data that is stored in the depot in the AccuRev repository, not data from any individual user's workspace.

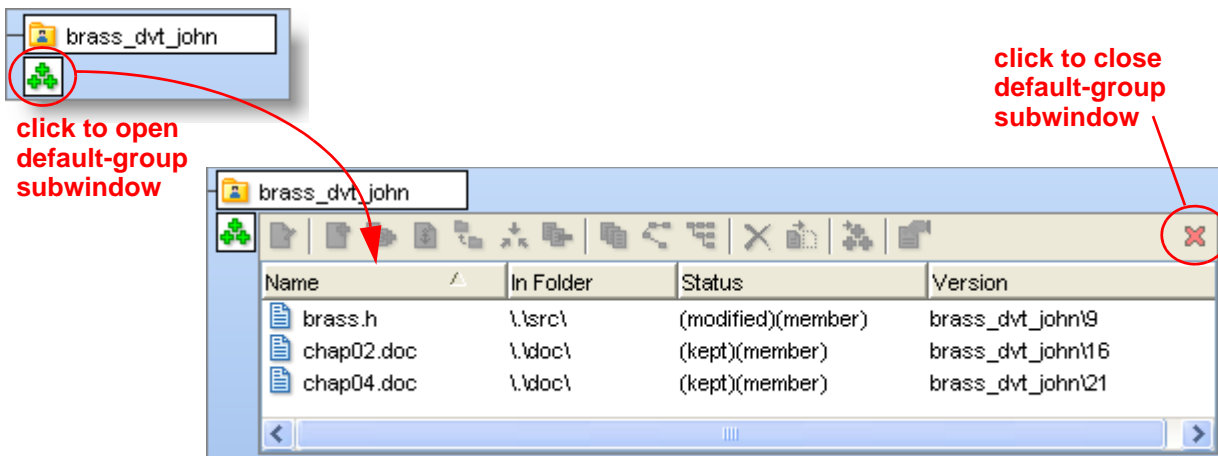
For more information, see *The File Browser* on page 43.




## Displaying and Working with the Default Group

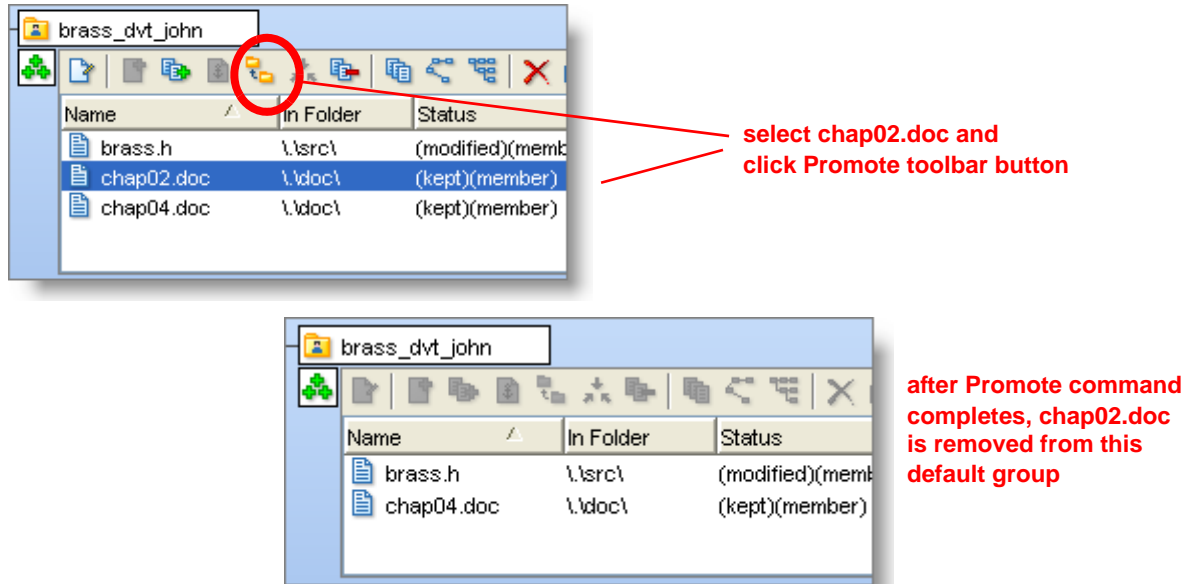
One of the most powerful features of the StreamBrowser is its ability to zero in on the “important” files — the ones that are under active development in a particular stream or workspace. AccuRev keeps track of “active elements” in each dynamic stream and workspace. Such elements are said to be in the default group of the stream or workspace.

Click the  control (the default group control) below a stream or workspace to open a subwindow that displays its default group. The control appears only if the default group is non-empty.




The default group subwindow closes when you click the “X” icon in its upper right corner. It also closes when you click a  control again — of the same stream or workspace, or of another one. (At most one default group subwindow can be open at any given time.)

The default group subwindow is small, but its scroll bars enable you to see any data that is not currently visible. It also has its own toolbar, similar to that of the File Browser's details pane. (Each element displayed has its own context menu, too.) This means you can perform many AccuRev operations in the default group subwindow, without having to invoke a File Browser at all. For example, you might promote file **chap02.doc** as follows:




At this point, you could open a window on the default group of the parent stream, to verify that **chap02.doc** is now active in that stream.

For another file, you might decide that it shouldn't have been edited in that particular workspace.

In that case, you could invoke the  **Revert to Backed** command in the default group subwindow to deactivate the file and remove it from the default group.

## Promoting the Entire Default Group

If you want to promote *all* the elements in a workspace's (or streams's) default group to the parent stream, you don't even need to open the default group subwindow. Instead of clicking the  default group control, drag-and-drop the control to the parent stream. The Promote dialog box appears, loaded with all the elements in the default group.

## Streams and Change Packages

The preceding sections discuss how the StreamBrowser and File Browser enable you to see what individual versions are present — and perhaps active — in a stream. AccuRev also makes it easy to answer questions like these:

“Are all the changes required to fix bug #4517 in this stream?”

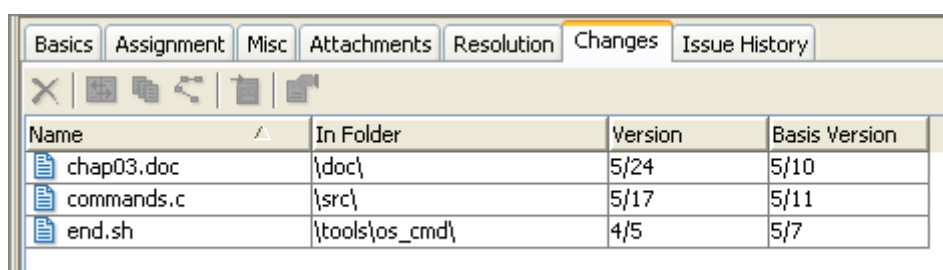
“Are any of the versions involved in the #4517 fix still active in this stream? (Or have all the changes already been promoted to a higher-level stream?)”

“What new features are still under active development in this stream?”

“The QA Group says they don’t have all the Color Mixer changes for the upcoming release. Is that true?”

The key is to go beyond thinking of individual versions to considering collections of versions, called change packages. With AccuRev, change packages are implemented by AccuWork issue records. An issue record records the details of a bug or feature: its description, how important it is, who originated it, who’s working on it, and so on. In the AccuRev Enterprise version of AccuWork, an issue record can also keep track of the changes that have been made to elements, in order to implement that particular bugfix or new feature.

For example, issue record #9 might contain a bug report, “Circles are not round”. The bugfix involves changes to three elements:



Name	In Folder	Version	Basis Version
chap03.doc	{doc\}	5/24	5/10
commands.c	{src\}	5/17	5/11
end.sh	{tools\os_cmd\}	4/5	5/7

change package  
for issue record #9  
contains three  
changes

When viewing the issue record through its edit form, go to the Changes tab to view the change package. In this example, the change package contains three changes:

- The changes that were made between version 5/10 and version 5/24 of element **chap03.doc**
- The changes that were made between version 5/11 and version 5/17 of element **commands.c**
- The changes that were made between version 5/7 and version 4/5 of element **end.sh**

For a discussion of how the head version (Version column) and basis version specifications define a change as a series of versions, see *Change Packages and Integrations between Configuration Management and Issue Management* on page 209 of the *AccuWork Issue Management Manual*.

Note: the Version and Basis Version columns always report real version-IDs — the IDs of versions as they were originally created in user’s workspaces — not the virtual version-IDs acquired as versions are promoted up the stream hierarchy.

## Change Packages “In” Streams

As the versions in a change package are promoted up the stream hierarchy, the change package itself implicitly moves up the hierarchy, also. Roughly speaking, a change package has risen to a certain level if all its entries have risen to that level. More precisely:

- Each change package entry is said to be “in” a particular stream if its head version (Version column) is the same as — or is an ancestor of — the version in the stream.
- A change package is said to be “completely in” a particular stream if all of its entries are “in”.

- A change package is said to be “partially in” a particular stream if some — but not all — of its entries are “in”.

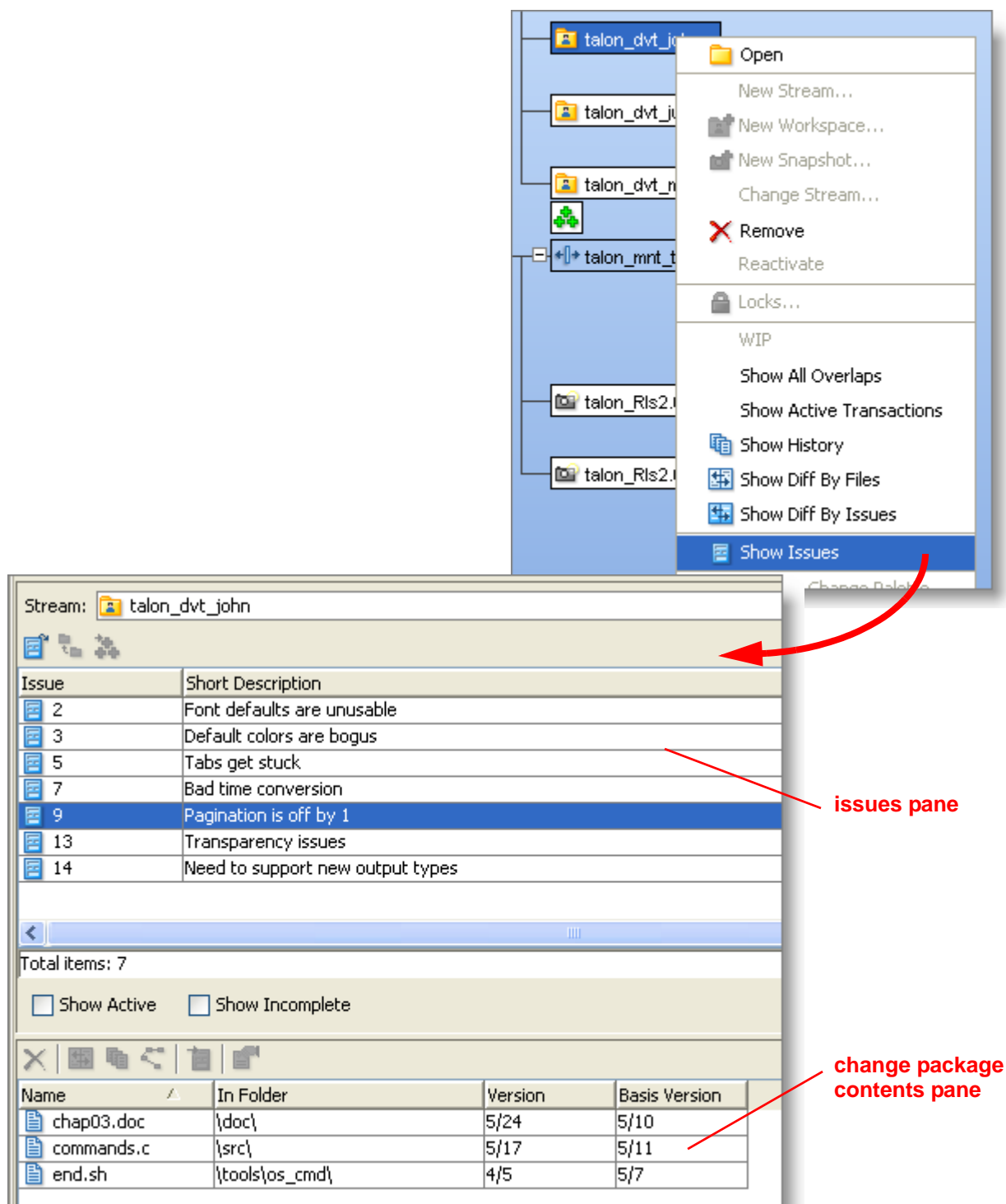
Two points need clarification here: why the “or is an ancestor of” clause? And why don’t we need to pay attention to the basis version, too?

The change package shows that versions 5/9 through 5/13 of **tools.readme** were created to fix the “Circles are not round” problem. Suppose that after that, some user brought version 5/13 of **tools.readme** into her workspace with an update, then created a new version in her workspace to fix another problem — say, version 7/3. Is the “Circles are not round” fix to **tools.readme** in version 7/3? Yes — because the newer version has version 5/13 as its ancestor, it’s safe to conclude that the changes in version 5/13 are still in version 7/3.

A similar ancestry analysis shows why we don’t need to worry about the base version in a change package entry. In this example, the change to **tools.readme** consists of the versions 5/9, 5/10, 5/11, 5/12, and 5/13. Head version 5/13 presumably contains all the changes in versions 5/9, 5/10, 5/11, and 5/12 — because it’s a direct descendant of those versions. So if 5/13’s changes are “in” a particular stream, it’s safe to conclude that the changes in 5/9 through 5/12 are in that stream, too.

## The ‘Show Issues’ Command

You can invoke the **Show Issues** command on any workspace, stream, or snapshot. (We’ll use “stream” to cover all three data structures in the remainder of this section.) This command opens a Stream Issues tab, which displays the change packages — that is, the issue records — that are “in” the stream, according to the definitions above.



The Stream Issues tab includes two panes, each with its own toolbar:

- The issues pane displays selected fields from the issue records whose change packages are “in” — partially or completely — the stream. The fields to be displayed are set in the Change

Package Results pane of the Change Packages subtab of the Schema Editor tab. Use the **Admin > Schema Editor** command to open a Schema Editor tab.

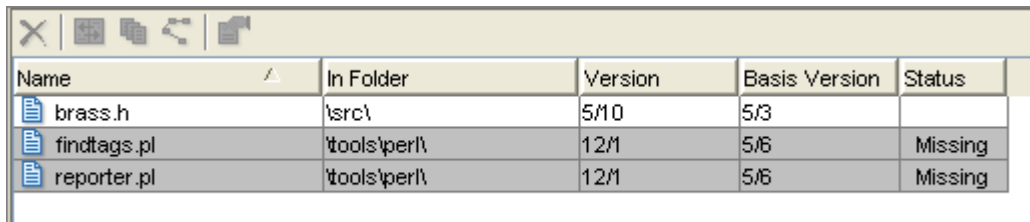
- The change package contents pane shows what’s in the change package of the currently selected issue record. This is identical to the issue record’s Changes tab.

Two checkboxes on the Stream Issues tab enable you to refine the change-package analysis:

### Show Incomplete

If **Show Incomplete** is cleared, the listing includes only change packages that are “completely in” the stream.

If the **Show Incomplete** is set, the listing includes only change packages that are “partially in” the stream. The Status column, along with background shading, indicates which change package entries are “in” the stream and which are “missing”.



Name	In Folder	Version	Basis Version	Status
brass.h	\src\	5/10	5/3	In
findtags.pl	tools\perl\	12/1	5/6	Missing
reporter.pl	tools\perl\	12/1	5/6	Missing

Note that the **Show Incomplete** checkbox is a toggle rather than a filter: setting and clearing the checkbox displays two non-overlapping collections of change packages.

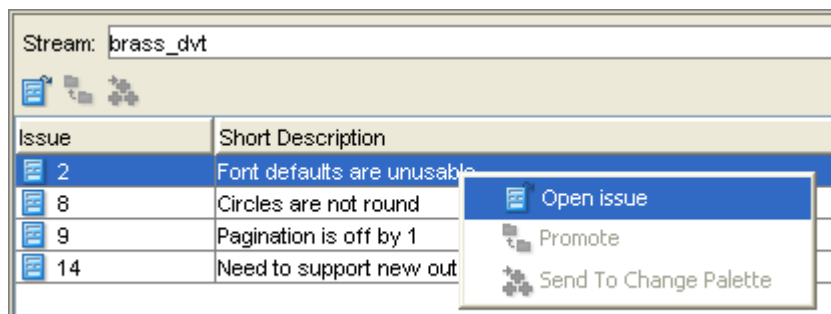
### Show Active

If **Show Active** is set, it filters the set of change packages displayed: a change package is included only if one or more of its versions is currently active in the stream (is in the stream’s default group). This filter helps you to concentrate on current programming efforts, rather than those that were completed long ago.

This filter has no effect when you invoke **Show Issues** on a snapshot, since no versions can be active in a snapshot.

## Working in the Issues Pane of the Stream Issues Tab

In the issues pane, you can select one or more issues to invoke commands on the issues, or on their change packages. These commands are available through context menus and in the toolbar of the issues pane.



### Open Issue

Open an edit form on the selected issue, and display the Changes tab.

## Promote

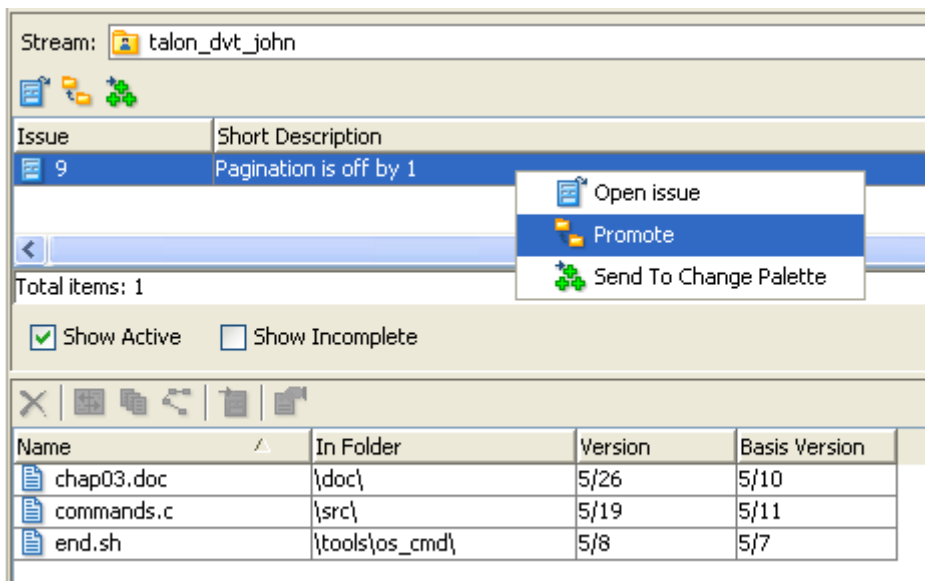
(enabled only if **Show Active** is set and **Show Incomplete** is cleared) Promotes each currently-active head version in the issue's change package to the parent stream. That is, if a version listed in the Version column is currently active in the stream on which you invoked the **Show Issues** command, that version is promoted to the parent stream.

## Send to Change Palette

(enabled only if **Show Active** is set and **Show Incomplete** is cleared) Opens a Change Palette tab, containing each head version in the issue's change package.

The following illustration shows the Stream Issues tab for workspace stream **talon\_dvt\_john**. Invoking the **Promote** command on issue #9 causes some or all these versions to be promoted:

version 5/26 of **chap03.doc**  
version 5/19 of **commands.c**  
version 5/8 of **end.sh**



The versions promoted are the ones that are currently in the workspace's default group.

## Working in the Change Package Contents Pane of the Stream Issues Tab

Each row in the change package contents pane displays an entry in the currently selected change package. You can select an entry and invoke any of the commands described below. In most cases the operation is performed on the entry's head version (Version column).

You can invoke the **Remove** and **Send to Issue** commands on a selection consisting of two or more change package entries.

## Open

Run the appropriate command on the version, according to its file type.

## View

Open a text editor on a temporary copy of the currently selected version (text files only).

## Save As

Copy the currently selected version to another filename.

## Remove

Remove the selected entry from the change package.

## Diff Against Basis

Compare the selected version with the corresponding basis version.

## Show History

Open a History Browser tab, containing the transactions involving the selected file or directory. See *The History Browser* on page 135.

## Browse Versions

Open a Version Browser tab, showing all the versions of the selected file or directory, and their interrelationships (ancestry). See *The Version Browser: Ancestry Tracking* on page 143.

## Send to Issue

Record the selected version(s) in the change package section (Changes tab) of one or more additional issue records. The default query of the issues database is executed, and you are prompted to choose one or more of the records selected by the query. You can also create a new issue record, to which the selected version(s) will be sent.

## Properties

Displays information about the selected element: pathname, file type of current version, and element-ID.

## Comparing the Contents of Streams


Another powerful StreamBrowser feature is its ability to quickly compare the contents of two streams. The **Show Diff By Files** command does *not* directly compare the contents of files. Streams contain versions of elements. So a comparison of streams might determine that one stream has version 5 of file **foo.c**, while the other stream has version 13. (It's likely that the two versions have different contents, but the stream-comparison command doesn't "look inside" the versions to find out.)

In addition, a stream comparison can report these differences:

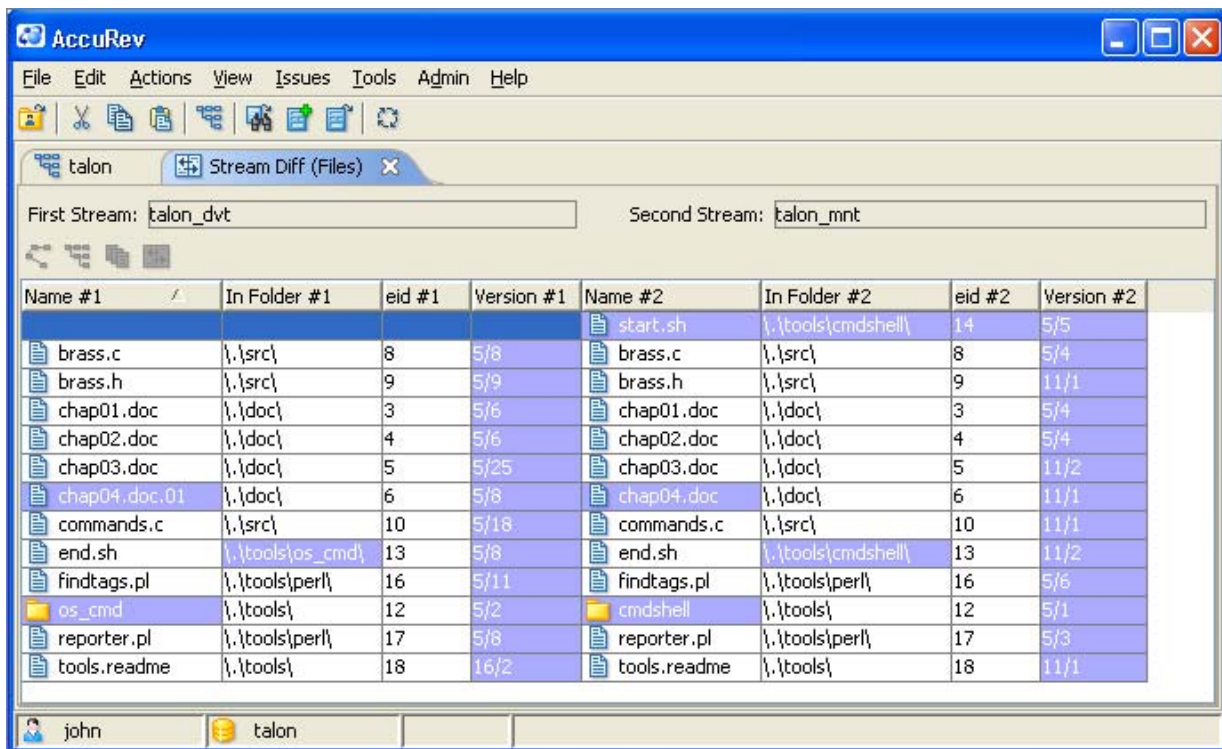
- An element has been renamed in one or both streams.
- A new element has been added to one of the streams (**Add to Depot** command)
- An element has been removed from one of the streams (**Defunct** command).

To compare two streams:




1. Select one of the streams and click the  **Show Files Difference** button on the StreamBrowser's toolbar. (Or select **Show Diff By Files** from the stream's context menu.) The selected stream is highlighted in green, and the "show differences" icon is added to the mouse pointer.
2. Click the other stream.

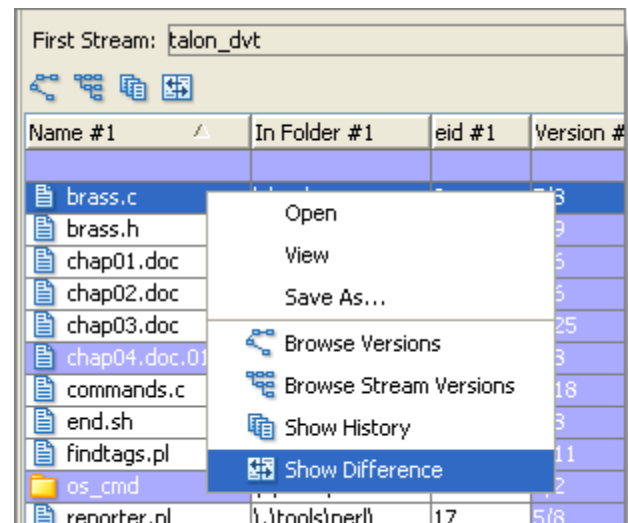
This opens a **Streams Diff** tab:



The example above shows a total of 13 differences between the two streams:

- One element, **start.sh**, appears in stream **talon\_mnt**, but not in stream **talon\_dvt**.
- One file has a different name in the two streams: **chap04.doc.01** vs. **chap04.doc**.
- One directory has a different name in the two streams: **os\_cmd** vs. **cmdshell**. This, in turn, causes file **end.sh** to have different pathnames in the two streams.

- Nine files have different versions in the two streams. For example, **talon\_dvt** contains version 5/8 of **brass.c**, but **talon\_mnt** contains version 5/5. To “drill down” and see what the changes are between these two versions, select the file and click the  **Show Difference** toolbar button (or right-click the file and select **Show Difference**).



You can invoke the following commands on versions listed in the Streams Diff tab, using the element’s context menu, the **Actions** menu, or the tab’s toolbar.

### Open

Run the appropriate command on the version, according to its file type.

### View

Open a text editor on a temporary copy of the currently selected version (text files only).

### Save As

Copy the currently selected version to another filename.

### Browse Versions

Open a Version Browser tab, showing all the versions of the selected file or directory, and their interrelationships (ancestry). See *The Version Browser: Ancestry Tracking* on page 143.

### Browse Stream Versions

Open a Stream Version Browser tab, depicting the versions that are currently each of the depot’s streams. See *The Stream Version Browser* on page 151.

### Show History

Open a History Browser tab, containing the transactions involving the selected file or directory. See *The History Browser* on page 135.

### Show Difference

Compare the contents of the file versions (text files only) that are in the two streams.


## Comparing Streams Using Change Packages

Section *Streams and Change Packages* on page 99 describes how you can determine what change packages are “in” a workspace, stream, or snapshot. Similarly, you can use the **Show Diff By Issues** command to cast the comparison of two workspaces/streams/snapshots in terms of change packages. (We’ll use “stream” to cover all three data structures in the remainder of this section.)

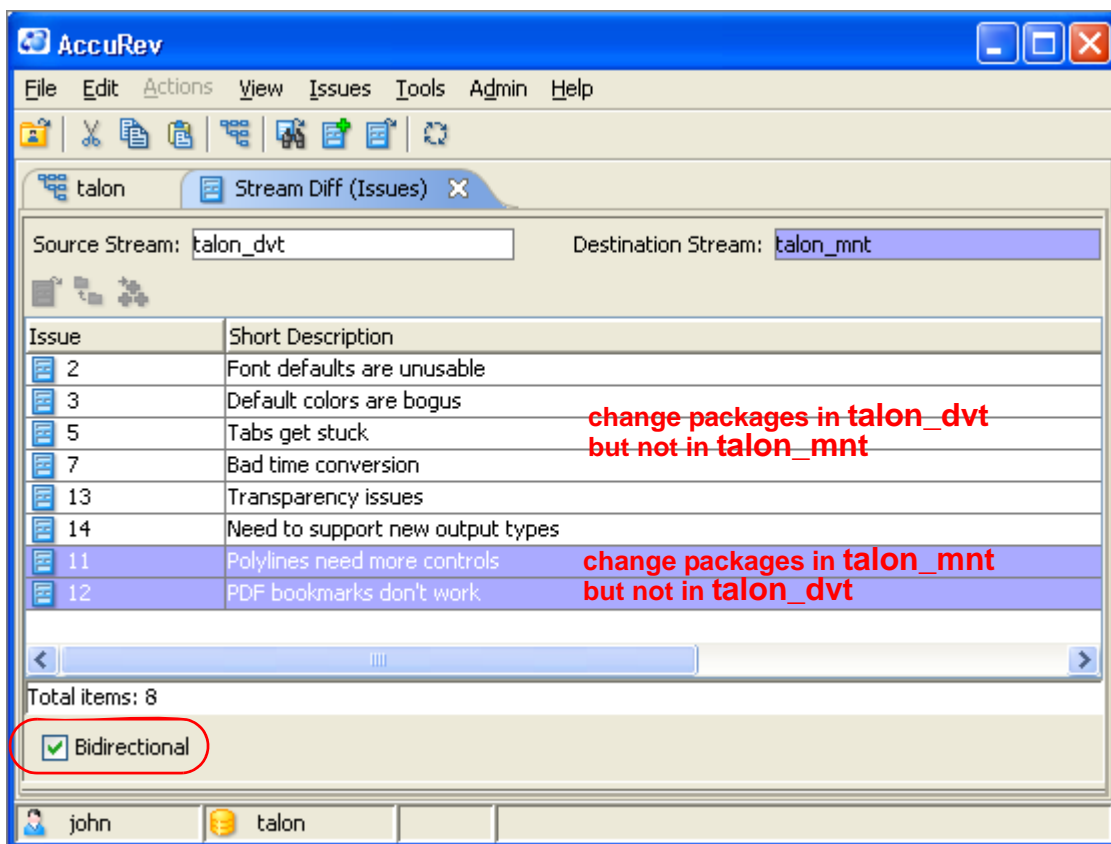
This enables you to easily determine the answer to such questions as:

“What features in the Integration stream have not yet been sent to the QA stream?”

To compare two streams by their change packages:

1. Select one of the streams and click the  **Show Issues Difference** button on the StreamBrowser’s toolbar. (Or select **Show Diff By Issues** from the stream’s context menu.) The selected stream is highlighted in green, and the “show differences” icon is added to the mouse pointer.
2. Click the other stream.

This opens a **Stream Issues** tab. Initially, this tab lists the change packages — that is, the issue records — that are in the first stream (“source”), but not in the second stream (“destination”). The names of the two streams appear at the top of the tab. If you check **Bidirectional**, the listing is expanded to show the change packages that are in the second stream, but not in the first stream. Shading is added to help you to distinguish the two sets of change packages.



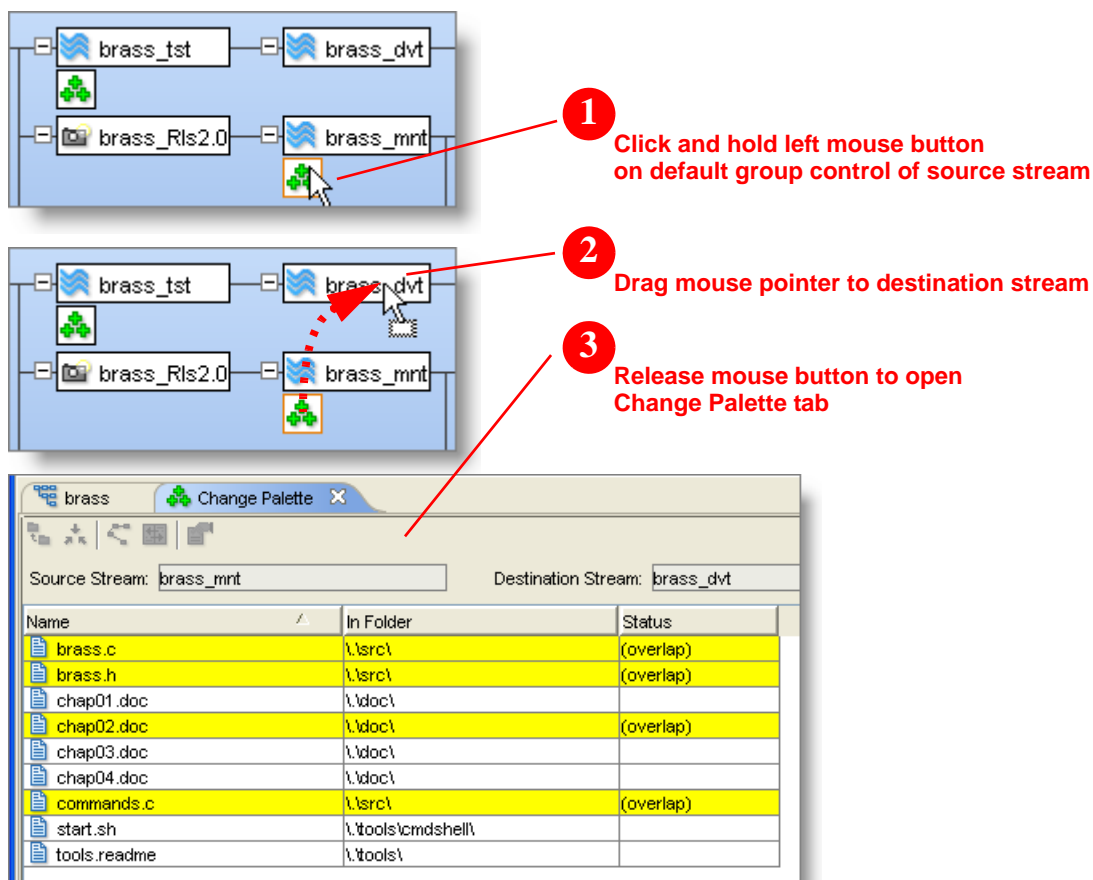
The **Show Difference By Issues** command considers only change packages that are completely “in” the specified streams. It does not consider change packages that would be displayed by the **Show Issues** command with **Show Incomplete** selected.

## Propagating Versions through the Stream Hierarchy

In *Displaying and Working with the Default Group* on page 98, we showed a simple example of how to use the StreamBrowser to promote files from a workspace or stream to its parent stream. More generally, AccuRev’s Change Palette enables you to promote versions directly between any two dynamic streams. The StreamBrowser makes it particularly easy to “load” versions into the Change Palette, so that you can then promote them:

### Propagating All of a Stream’s Changes

You can use a drag-and-drop operation to propagate all of a stream’s changes to a stream that is not the direct parent. This is termed a cross-promotion (rather than simple promotion) of the versions. The steps are illustrated below.



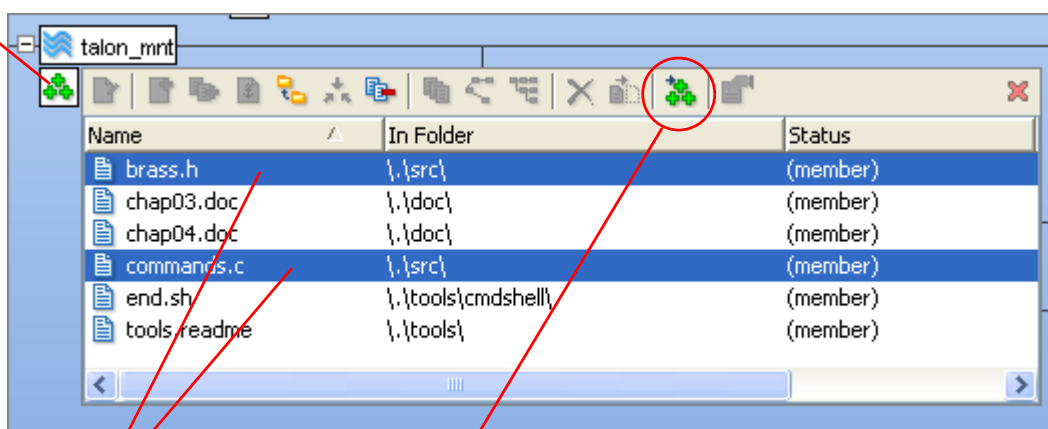
This opens a Change Palette tab, loaded with all versions in the source stream containing changes that have not yet been sent to the destination stream. For information on what to do next, see *Using the Change Palette* on page 155.

## Propagating Selected Changes from a Stream

Sometimes you want to send just some of a dynamic stream's changes — not all of them — to another stream. To load one or more versions in a stream's default group into the Change Palette:

1. Open the dynamic stream's default group subwindow. Alternatively, open a File Browser tab for that dynamic stream, and execute one of the searches that displays active elements — for example, the **Default Group** or **Overlap** search.
2. Select one or more elements in the resulting display.
3. Execute the **Send to Change Palette** command from the toolbar, or from elements' context menu.

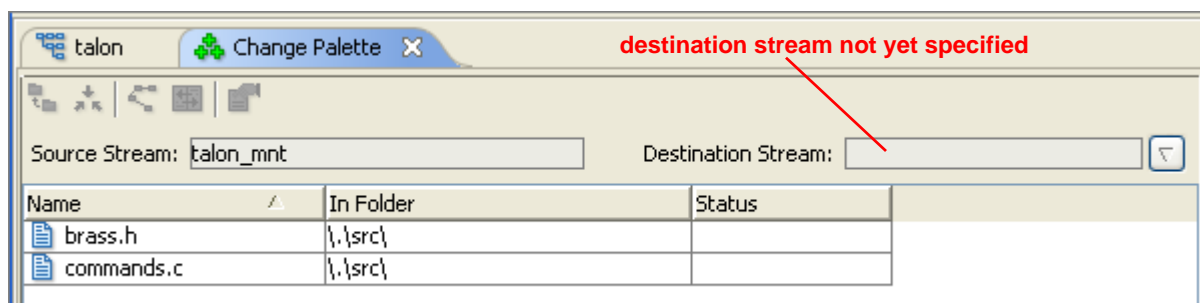
1. Open stream's default group subwindow



2. Select elements

3. Execute Send To Change Palette command

This places the selected version(s) in the Change Palette. Note that this procedure specifies the source stream, but you still need to specify the destination stream.



For information on what to do next, see *Using the Change Palette* on page 155.

## Additional Operations on Streams, Snapshots, and Workspaces

In addition to the operations described in the preceding sections, the StreamBrowser provides commands that operate on a selected stream, snapshot, or workspace. These commands are available through the StreamBrowser toolbar and through the context menu of a selected object.

This section describes all these commands. Some of them are not valid in all contexts, as noted below.

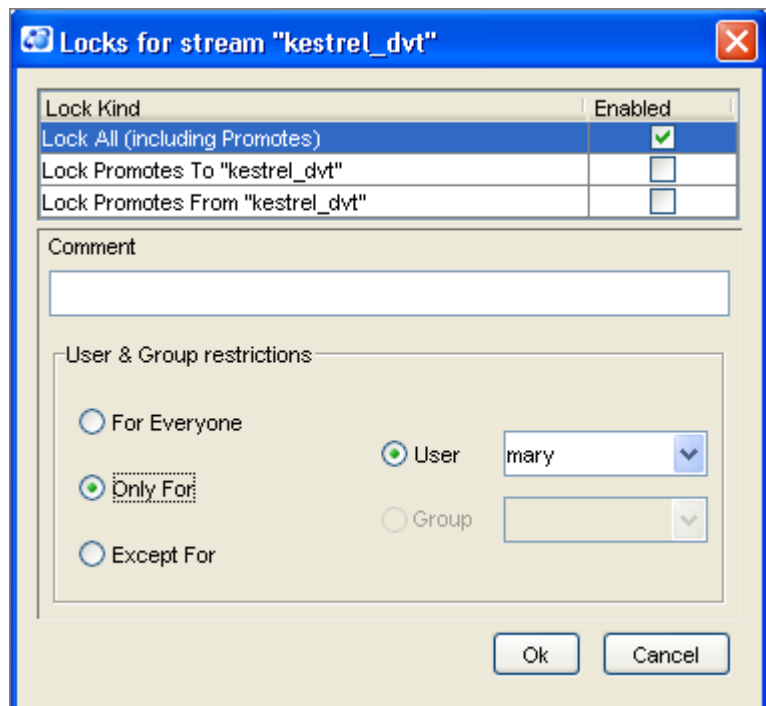
### Locks ... (Lock/Unlock Stream)

(Dynamic stream only)  
Locking a stream disables various operations that modify the stream. The **Lock All** choice disables promotion of versions to/from the stream, disables include/exclude mode changes to the stream, and disables the **Change Stream** command for the stream. With **Lock Promotes To**, you can disable promotion to the stream; With **Lock Promotes From**, you can disable promotion from the stream.

(A **Lock All** setting takes precedence over “to” and/or “from” settings.)

You can restrict the effect of the lock to particular users or groups; alternatively, you can exempt particular users or groups from being affected by the lock.

If a stream is already locked, a lock icon appears in the StreamBrowser graphical display, and in the Locks column of the tabular display.



### WIP (Work in Progress)

Displays the contents of the default group of every workspace backed by (directly below) the specified stream or snapshot.

### Show All Overlaps

(Workspace only) For active file elements (those in the workspace’s default group), display all overlaps involving versions in the workspace’s entire backing chain (higher-level streams) — not just overlaps between the workspace and its immediate backing stream.

### Show Active Transactions

(Dynamic stream only) Invoke a History Browser, containing the transactions that created the versions currently in the stream’s default group. A transaction is listed if *any* — not necessarily all — of its versions are currently active in the stream.

In certain cases, this command does not report a transaction that you might expect to be listed. An active version might not *need* to be promoted to its parent, because one of its descendant

versions has already been promoted there. This situation is called an “underlap”. Such a version is ignored by this command’s determination of which transactions are currently active.

### Show Issues

Analyzes the change packages stored in the depot’s AccuWork issues database. Reports on change packages that have been (partially or completely) incorporated into the stream.

### Show Diff By Issues

Prompts you to indicate another stream. Reports on differences between your stream and the other stream, in terms of change packages instead of individual elements.

### Show History

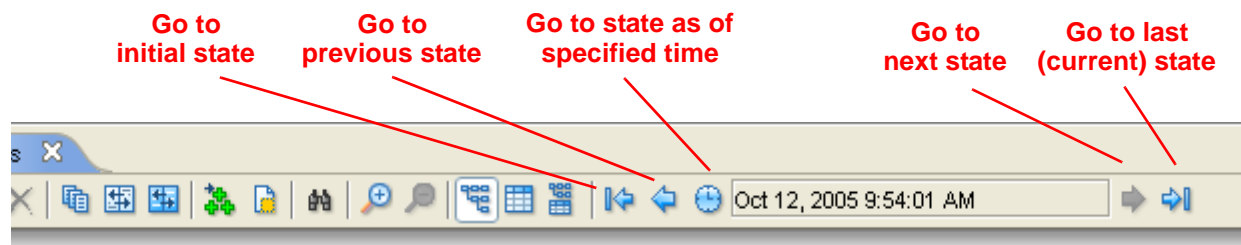
Invoke a History Browser, containing every transaction that created a version in the stream.

### Show Patch List

Prompts you to indicate another stream. Lists all the individual versions that need to be patched to that other stream, in order to have the other stream include all the changes in your stream.

## Stream History

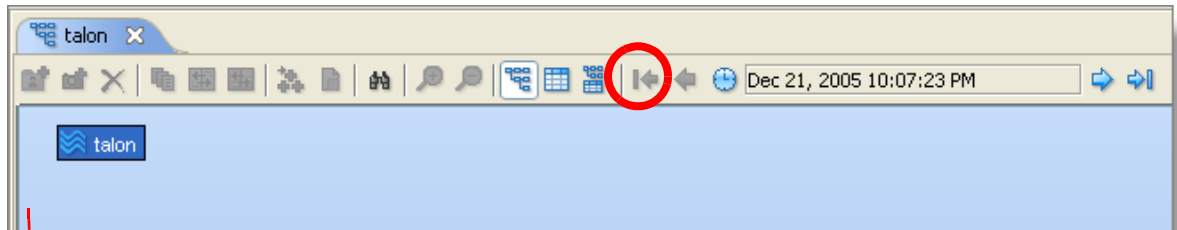
Like the individual files under version control, a depot’s stream hierarchy undergoes change, too. Initially, the depot contains a single “base” stream. Thereafter, additional dynamic streams are created, along with snapshots and user workspaces. Each such change in the stream hierarchy creates a new “state” of the hierarchy. You can browse through all these states, like browsing through a slide show, using the History controls in the StreamBrowser’s toolbar:



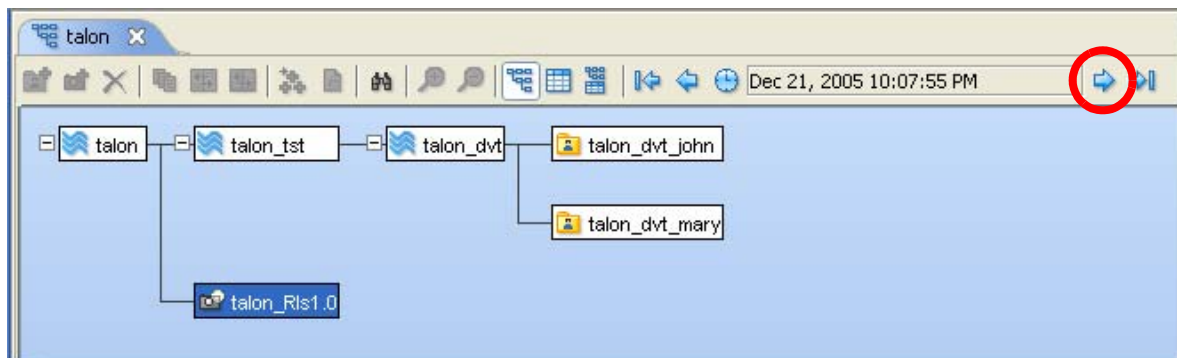
Note: These controls appear only if you’ve checked the **Enable Stream Browser History** preference. If you change the preference after opening a StreamBrowser tab, use **View > Refresh** to make the controls appear (or disappear). See *User Preferences* on page 6.

Here's a simple example:

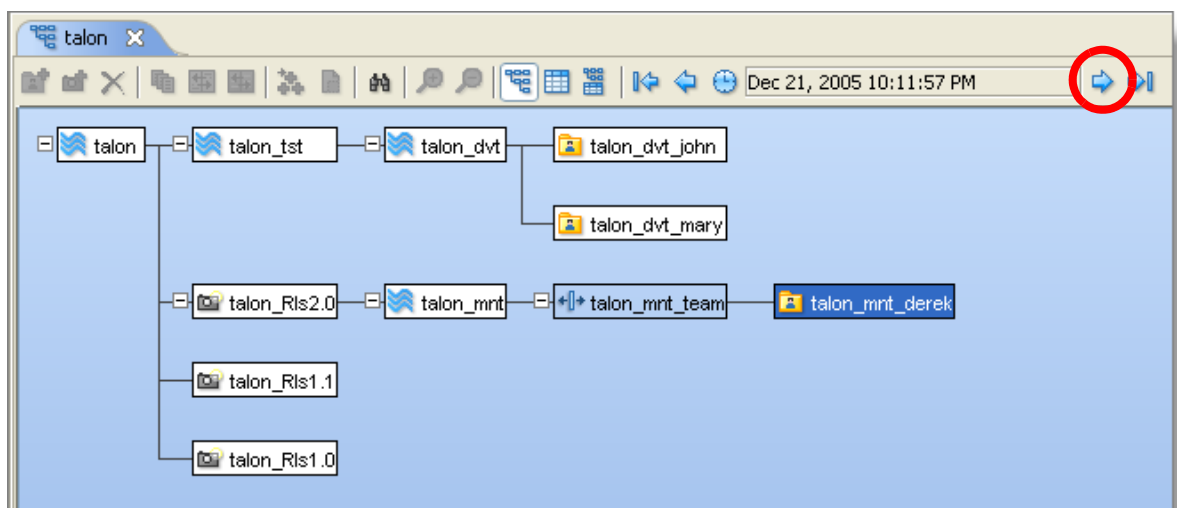
**1** click  to return to first state



**2** click  a few times




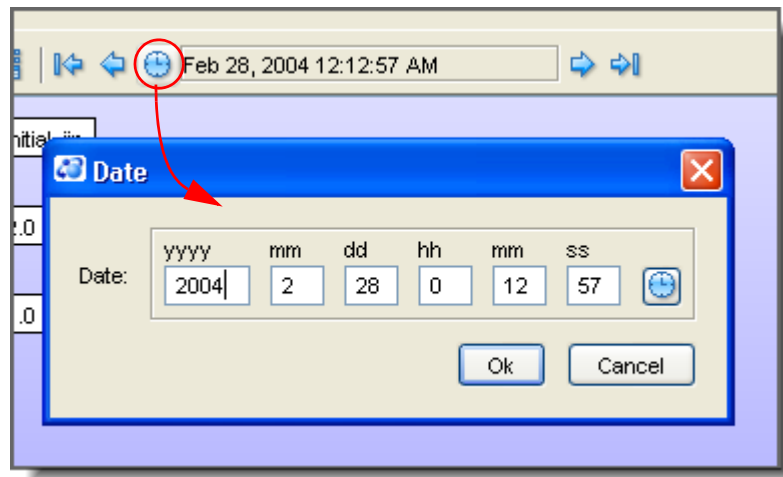
**3** click  a few times more





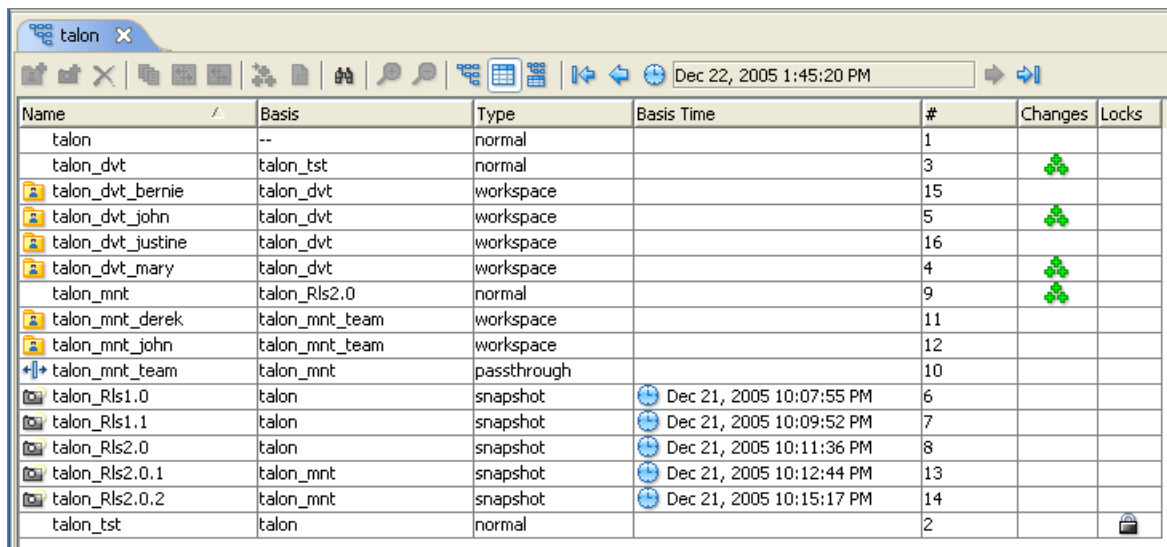
This “slide show” is read-only. You cannot select or operate on any of the displayed streams until you return to the current state of the stream hierarchy.
























When you click the  clock toolbar button, a dialog box appears in which you can specify a particular time. This causes the “slide show” to jump to the state that existed at that time.



## Tabular StreamBrowser Display

In tabular mode, the StreamBrowser tab displays a table listing some or all of the current depot’s streams. If you have used the **Zoom In** command in graphical mode to restrict the display to a subhierarchy, only those streams appear in the table. (You can also invoke the **Zoom In** and **Zoom Out** commands in tabular mode.)



Name	Basis	Type	Basis Time	#	Changes	Locks
talon	--	normal		1		
talon_dvt	talon_tst	normal		3		
 talon_dvt_bernie	talon_dvt	workspace		15		
 talon_dvt_john	talon_dvt	workspace		5		
 talon_dvt_justine	talon_dvt	workspace		16		
 talon_dvt_mary	talon_dvt	workspace		4		
talon_mnt	talon_Rls2.0	normal		9		
 talon_mnt_derek	talon_mnt_team	workspace		11		
 talon_mnt_john	talon_mnt_team	workspace		12		
 talon_mnt_team	talon_mnt	passthrough		10		
 talon_Rls1.0	talon	snapshot	 Dec 21, 2005 10:07:55 PM	6		
 talon_Rls1.1	talon	snapshot	 Dec 21, 2005 10:09:52 PM	7		
 talon_Rls2.0	talon	snapshot	 Dec 21, 2005 10:11:36 PM	8		
 talon_Rls2.0.1	talon_mnt	snapshot	 Dec 21, 2005 10:12:44 PM	13		
 talon_Rls2.0.2	talon_mnt	snapshot	 Dec 21, 2005 10:15:17 PM	14		
talon_tst	talon	normal		2		

The checkboxes and list box at the bottom of the tab work the same way as in graphical mode to control which streams are included in the table. As with most AccuRev tables, you can define single-column or multiple-column sorts to reorder the table’s rows. You can drag column separators and drag-and-drop columns to rearrange the table’s columns.

The StreamBrowser’s tabular mode offers the same command toolbar and context menus as the graphical mode.

# The AccuRev Diff, Merge, and Patch Tools

The AccuRev GUI includes tools that process two versions of the same file element:

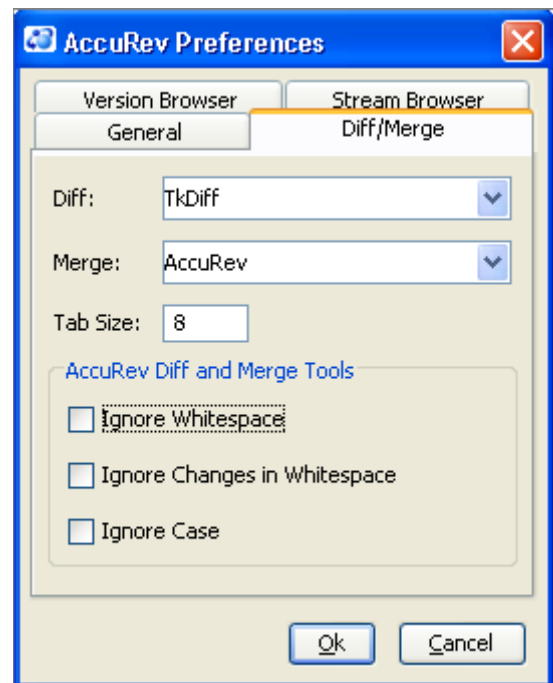
- The **Diff** tool compares any two versions of a file element.
- The **Merge** tool combines the changes in two versions of a file element: your workspace's version and the version in the stream to which you intend to promote the workspace version.
- The Merge tool can also be used to perform a **Patch** command (patch operation instead of merge operation). This enables you to incorporate just the recent changes made in the non-workspace version.

Since the merging of two versions of a file is a logical extension of finding their differences, the Diff and Merge tools are similar in many respects.

## Enabling the Diff and Merge Tools

By default, the AccuRev GUI uses the “AccuRev” native Diff and Merge tools. But you can configure it to use the tools that were included in previous AccuRev releases, or to use third-party or home-grown tools. The **Tools > Preferences** command brings up a dialog box with a Diff/Merge tab:

These configuration settings are persistent; they are automatically reestablished the next time you start the AccuRev GUI.



Instead of selecting one of the Diff or Merge tools in the drop-down listing, you can enter a tool-invocation command line in the input field. This command line must include substitution patterns, as indicated in the following table:


	Pattern	Meaning in Command String
Custom <b>Diff</b> Tool	%1%	first version to be compared
	%2%	second version to be compared
Custom <b>Merge</b> Tool	%a%	common ancestor file
	%1%	version in backing stream
	%2%	file in workspace
	%o%	merge results (output) file

Make sure the alternative diff or merge program is located in a directory on your search path. The exit status of the merge program should be:

- Zero if there are no conflicting changes. The **merge** command's default "do next" action after running this program will be **keep**.
- Non-zero if there are conflicting changes. The **merge** command's default "do next" action after running this program will be **edit**.

## Invoking the Diff Tool

There are multiple ways to specify the two versions of a file element to be compared with the Diff tool:

- In a File Browser, each file's context menu includes a **Diff Against...** item. There are several choices, each of which compares your version (i.e. the file in your workspace) with a version stored in the depot:
  - **Most Recent Version:** Compares your file with the version currently in your workspace stream. Use this choice if you've modified the file since the last time you performed a **keep** on it (or if you've never performed a **keep** on it since the last update).
  - **Backed Version:** Compares your file with the version currently in the backing stream. For example, you might use this choice to see *all* the changes you've made to this file since you updated your workspace and starting working on the file. (And assuming no one else has promoted a new version to the backing stream in the meantime.) This might include the changes stored in several intermediate versions that you've created with **keep**.
  - **Basis Version:** Compares your file with the version that you started working with, before making your "recent" changes. For a discussion of the meaning of "recent", see *Structure of a Change Package* on page 209 of the *AccuWork Issue Management Manual*.
- In a StreamBrowser display, view the elements in a stream's default group by clicking the  control below the stream. An element's context menu includes the **Diff Against...** items described above.

- In a Version Browser or Stream Version Browser display for a text-file element, you can compare any two versions: right-click any version, select **Diff Against Other Version** from the context menu, then left-click any other version.
- In a History Browser display for a text-file element, you can compare the versions created in any two transactions. Right-click a transaction in the summary (upper) pane, select **Diff Against Other Version** from the context menu, then left-click another transaction in the summary pane. Similarly, the **Diff Against File in the Workspace** command enables you to compare the version created in a particular transaction with the file currently in your workspace. (That file might contain changes that you haven't yet preserved with **Keep**).

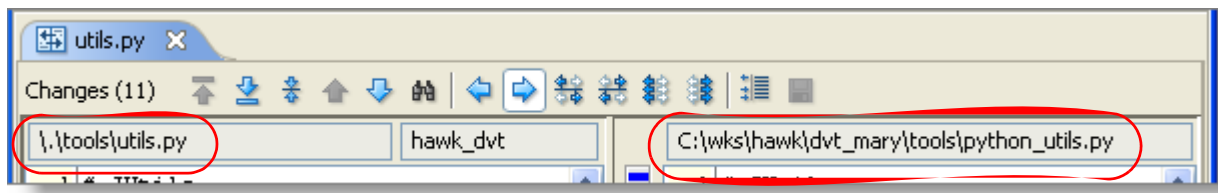
## Using the Diff Tool

Each time you invoke it, the Diff tool opens a new tab of the overall AccuRev GUI window. When you're finished comparing the versions, you can close the Diff tab like any other tab.

Diff displays the two specified versions side-by-side, as described in the following sections.

### Difference in Pathname

The two versions being compared can have different pathnames: the element may have been renamed or moved to another directory within the depot. (This may have occurred in one of the versions, or in both of them.) The Diff tool does not highlight or announce such a namespace-level difference. It shows the pathname of each version at the top of the difference display, enabling you to quickly determine whether they differ.



### Comparison of Binary Files

If the versions to be compared are in a binary image format that AccuRev can render, the Diff tool simply displays the versions, so that you can determine their differences by inspection. AccuRev can render the following image formats:

- JPEG (.jpg or .JPG filename suffix)
- PNG (.png or .PNG)
- GIF (.png or .GIF)

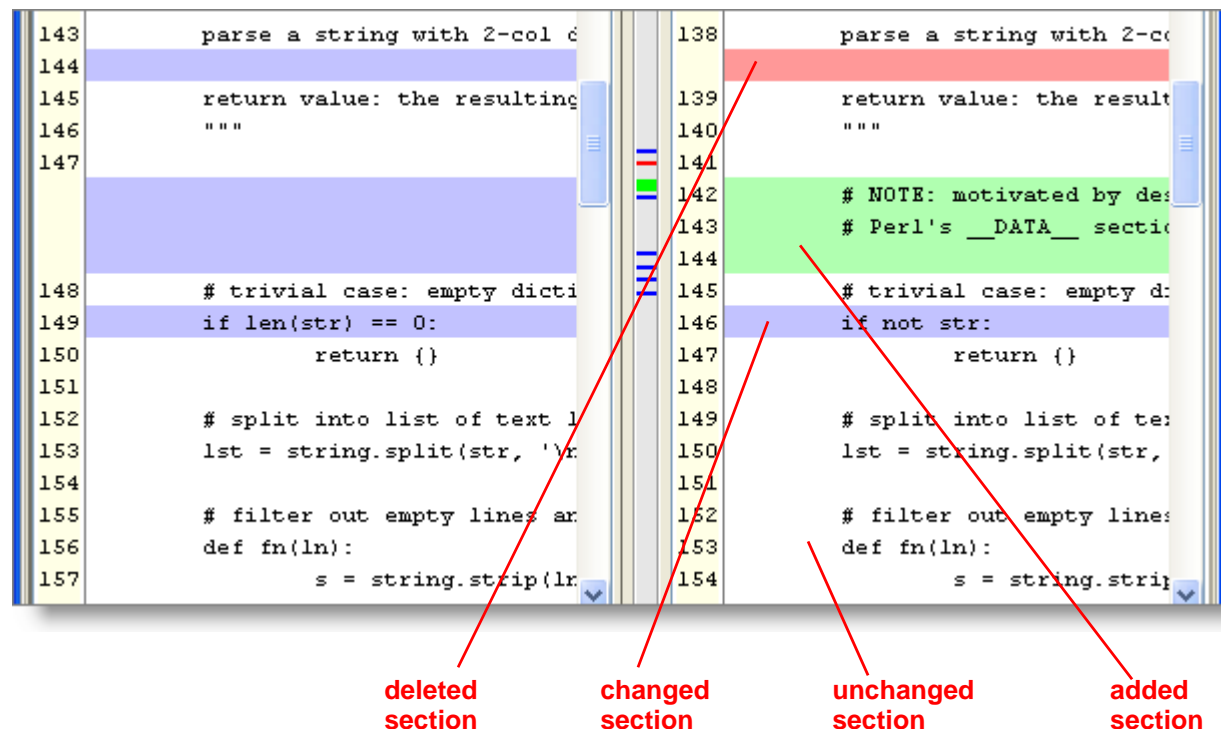
### Comparison of Text Files

If the versions to be compared are text files, the Diff tool displays them side-by-side in separate panes, so that corresponding text lines in the versions line up visually. We'll call them the "before version" (displayed on the left) and the "after version". Initially, the two panes are the same width,

but you can drag the vertical separator to change the relative widths. To make both panes wider, just increase the size of the overall AccuRev GUI window.

Note: depending on how you launch the Diff tool from the Version Browser, an older version might be displayed on the right, not the left. We recommend keeping the older version on the left, so that “before” and “after” in the descriptions below correspond to reality.

Through color-coding, the Diff tool partitions the text into the following kinds of sections:



- **Unchanged Sections:** Text sections in which the versions are identical are displayed with a **white** background.
- **Added Sections:** If a text section occurs only in the “after” version, it is displayed there with a **green** background. Empty space, colored blue but without any line numbers, appears at that point in the “before” version.
- **Deleted Sections:** If a text section occurs only in the “before” version, empty space with a **red** background appears at that point in the “after” version. The deleted section is colored blue in the “before” version.
- **Changed Sections:** Sometimes, the Diff tool decides that a text section in the “before” version has been revised, producing the corresponding section in the “after” version. The before and after sections are not necessarily the same length, in which case some empty space is displayed in one of the versions. Both the before and after sections appear with a **blue** background.

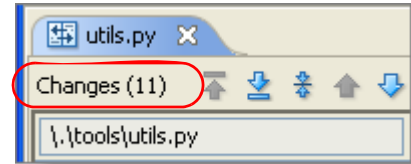
If a text block has been moved from one location in the file to another, the Diff tool indicates this as two separate changes: a section deleted at the original location, and another section added at the new location.

## Navigating the Differences



Most text files are too long to fit on the display screen, and some are too wide. Accordingly, the panes in which the versions appear have both vertical and horizontal scroll bars. When you scroll one of the panes in any direction, the other one scrolls automatically. Line numbers at the edge of each pane help you to keep oriented as you scroll through long files.

For a text file that contains hundreds or thousands of lines, there may be only a few difference sections (added, deleted, or revised), separated by large unchanged sections. The Diff tool offers two ways to navigate among the difference sections:

- **Sequential Access:** There's a status indicator and a toolbar at the top of the Diff tab. (This is distinct from the toolbar for the overall AccuRev GUI window.) The status indicator shows of the total number of difference sections found.



The toolbar buttons  **Next Diff** and  **Previous Diff**

provide sequential access to all the difference sections. The  **First Diff** and  **Last Diff** buttons are useful for restarting a scan of all the difference sections. You can use the following keyboard accelerators instead of these buttons:

**Ctrl-F** go to first difference section

**Ctrl-L** go to last difference section


**Ctrl-P** go to previous difference section

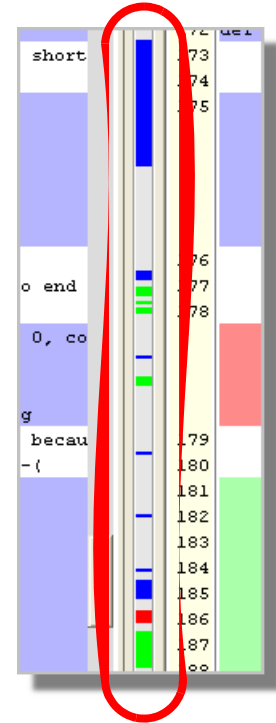
**Ctrl-N** go to next difference section

Whenever you use one of these buttons to jump to a particular difference section, the Diff tool remembers it as the current difference and highlights the lines numbers in both panes.


- **Random Access:** Between the two panes, there's a difference map that shows the relative locations and sizes of all the difference sections. The map uses the same color-coding as the difference sections themselves: added sections in green, deleted sections in red, revised sections in blue. Click on any colored area of the map to scroll both panes directly to the corresponding difference section. (Actually, you can click *anywhere* in the difference map; the panes will scroll to that location, even if the files are identical there.)

Using the difference map merely scrolls the panes; it does not change the current difference. But after you jump to a particular location, you can then click the difference section at that location to make it the current difference.

Using the scroll bars or the difference map, you can make the current difference scroll offscreen. To bring it back onscreen, click the  **Center** button on the Diff tab's toolbar.




## Searching for Text

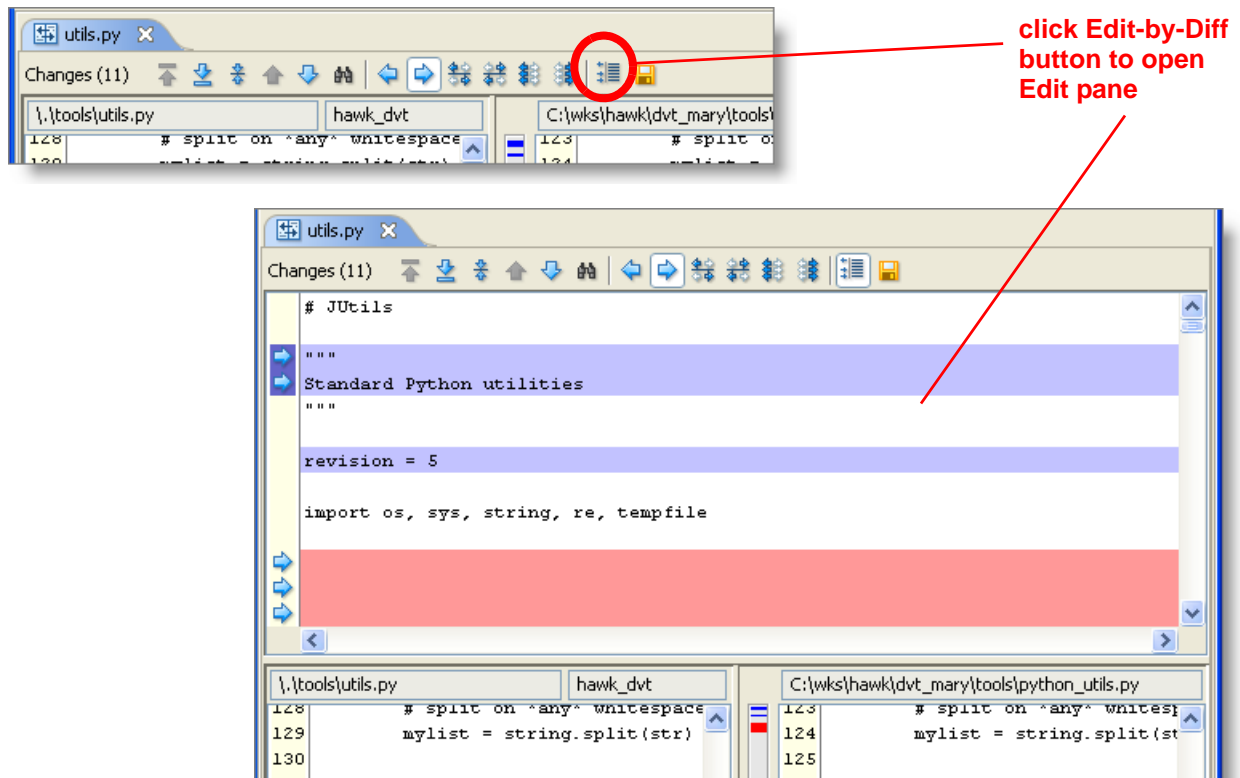
In addition to navigating among the difference sections, you can search for any text string in either pane, using the  **Search** toolbar button — or the **Ctrl-S** keyboard accelerator.










## Editing a File Using the Diff Tool

In addition to comparing two versions of a file, the Diff tool can help you to edit the contents of the version in your workspace. This Edit-by-Diff capability is available only when you're comparing your workspace version with another version; it's not available when you invoke the Diff tool from the Version Browser.

Here's a procedure for using this capability:

1. Invoke the Diff tool in a File Browser, to compare your workspace version with some other ("before") version of the same file. (You cannot invoke Edit-by-Diff from the Version Browser.)
2. Click the  **Edit-by-Diff** button in the Diff toolbar. This opens a third pane (Edit pane), which initially contains the same text as your workspace version.



3. As you browse through the difference sections, using the techniques described above, you can click the  **Revert My Change** toolbar button to swap in the “before” version’s text at that point (a change, addition, or deletion). You can also add the “before” text to your text, using one of the “take both” buttons,  or .
4. At any time, you can edit text manually in the Edit pane.
5. If you change your mind about a difference section where you’ve swapped in text from the “before” version, select that section and click the  **Restore My Change** toolbar button. Any manual edits you’ve made in that difference section will be lost.
6. Clicking the  **Revert All of My Changes** button is a shortcut for visiting every difference section and clicking  **Revert My Change**. Similarly, the  **Restore All of My Changes** button is a shortcut for visiting every difference section and clicking  **Restore My Change**.
7. When you’re done, click the  **Save Edits & Close** toolbar button. This simply replaces the file in your workspace. (It doesn’t perform an AccuRev **keep** or **promote** command.) To cancel the edit session without saving any changes to the file, just close the Diff tab.

Note: using the Edit-by-Diff capability is similar to using the Merge tool (described below). An important difference is that an Edit-by-Diff operation involves just two versions, and a Merge operation takes into account a third version: the closest common ancestor of the two versions being merged.



## When to Use the Merge Tool

Perhaps the most common usage pattern in AccuRev is:

- Using **keep** to create one or more versions of a text file in your workspace.
- Using **promote** to propagate the most recently kept version to the backing stream.

Occasionally, someone else “gets there first”. That is, both you and a colleague are working on the same file concurrently, each of you using a copy of the file in your own workspace. And the colleague promotes his changes to the common backing stream before you do.

Before you can promote your own changes to the backing stream, you must first merge the changes from your colleague’s version with your changes. Merging ensures that no one’s work is inadvertently lost or overwritten in a concurrent development environment.

The AccuRev GUI makes it easy to tell which files require a merge prior to promotion. If a file needs to be merged, the details pane in your workspace shows the file as having an **overlap** status — that is, the work of one or more of your colleagues has overlapped your own. If you select **Overlap** in the File Browser’s searches pane, the details pane displays all your **overlap**-status files: all files in the depot that you must merge before you can promote them from your workspace to the backing stream.

Although the above scenario is by far the most common one, AccuRev’s flexibility allows for other merge scenarios, too:

- You can modify the file in your workspace by “pulling in” the changes from *any* stream’s version of the file, not just the backing stream’s version.
- You can merge any two dynamic streams’ versions of the file.

These scenarios involve using the GUI’s **Change Palette**. In all cases, however, you use the Merge tool in the same way, to combine the contents of two versions of the same file element.

## The Merge Algorithm

Before describing the Merge tool’s interface, we briefly describe how the tool works.

The Merge tool analyzes two contributor versions of a file, at both the content level and the namespace level:

- It combines the two versions’ contents to produce a merged version of the file. Sometimes, it can produce the merged version completely automatically; other times, it needs help from you to resolve conflicts between the versions.
- It compares the pathnames of the two versions. If the pathnames differ — because one or both of the contributors was renamed, or was moved to a different directory within the depot — it applies the change to the merged version. As above, the Merge tool sometimes makes the name change automatically, but sometimes needs help from you to resolve a naming conflict.

The analysis that the Merge tool performs on the contributor versions at the content level is similar to that performed by the Diff tool. Both tools identify difference sections, where the contributors differ from each other. That’s all the Diff tool needs to do, but the Merge tool goes on

to decide how each difference represents a change from the past. In this context, “the past” means the closest common ancestor version of the two contributor versions.

To emphasize the Merge tool’s perspective, we use the term change section to describe a location where the contributors differ from each other. (In the context of the Diff tool, we use the term difference section.)

## A Non-Conflicting Content Change

For example, suppose a change section consists of 13 lines that occur in contributor #2 but not in contributor #1. To determine what kind of change this represents, the Merge tool looks at the corresponding location in the closest common ancestor version:

- If those 13 lines exist in the ancestor version, the Merge tool concludes that a change was made in contributor #1 (the lines were deleted) but no change was made in contributor #2 (the lines are still there).
- If the 13 lines do not exist in the ancestor version, the Merge tool concludes that a change was made in contributor #2 (the lines were added) but no change was made in contributor #1 (nothing was added).

In both these cases, there was a change from the common ancestor in exactly one of the contributors. The Merge tool deems this a non-conflicting change. It automatically incorporates the change (be it an addition, a deletion, or a revision of existing text) into the merged version.

## A Conflicting Content Change

Let’s take another example. A one-line error message has a slightly different wording in the two contributors:

- Contributor #1:  

```
#define E_COLOR498      "No color with that name was found."
```
- Contributor #2:  

```
#define E_COLOR498      "Color name unknown."
```

The following line occurs at the corresponding location in the closest common ancestor version:

```
#define E_COLOR498      "Huh?"
```

In this situation, the Merge tool finds a change from the common ancestor in *both* contributors, not just one of them. This is a conflicting change (or more simply, a conflict). The Merge tool doesn’t try to decide which contributor’s change is better. It just makes it easy for *you* to make this decision when you perform the merge.


Note: there’s another kind of change, where both contributors have made *the same* change from the common ancestor version. For example, both contributors might have replaced the error message **Huh?** with the message **No such color**. In such cases, the Merge tool silently incorporates the agreed-upon change into the merged version.

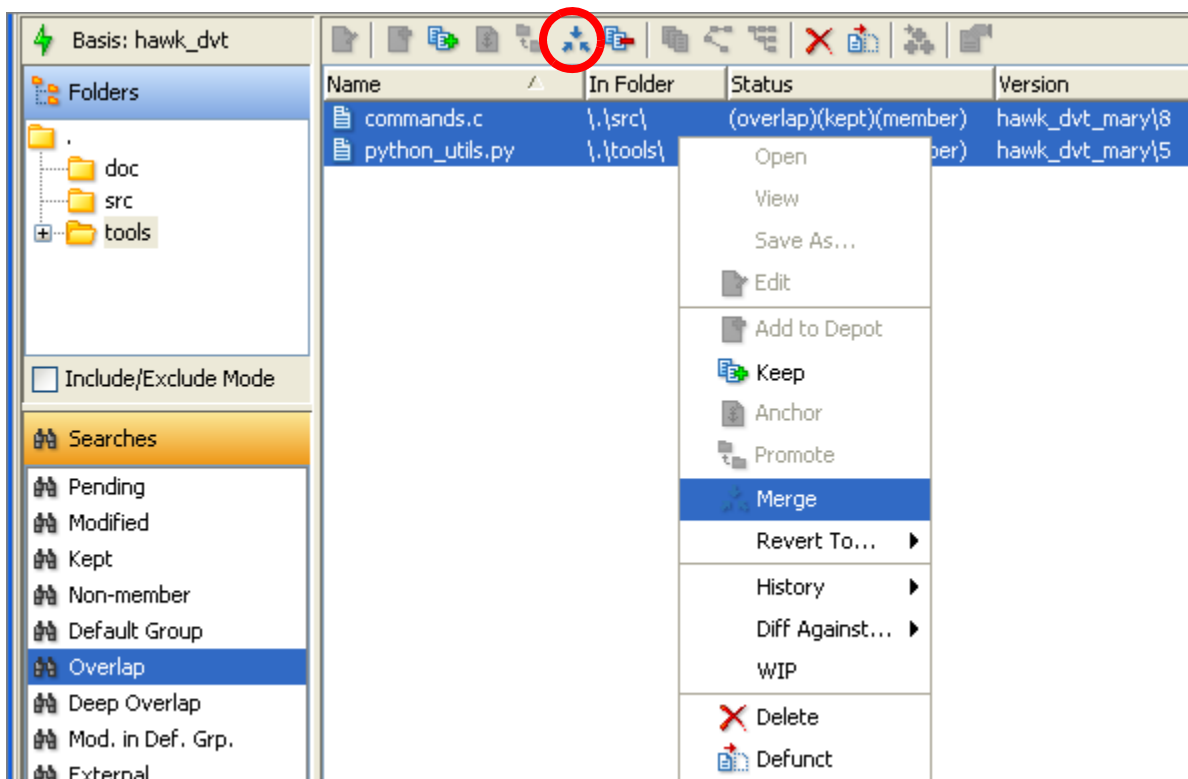
## Changes at the Namespace Level

Performing a merge at the namespace level is simpler than at the content level:

- If exactly one of the contributors had its pathname changed, the merged version automatically gets the new pathname.
- If both contributors had their pathnames changed, the Merge tool prompts you to select one of the new pathnames. (You can also choose to go back to the original pathname of the common ancestor version.) The merged version gets the pathname you choose.
- If both contributors had their pathnames changed in exactly the same way — for example, both versions were renamed from **chap01.doc** to **chapter01.doc** — the merged version automatically gets that new pathname.

## Invoking the Merge Tool

To invoke the Merge tool, select one or more files with **(overlap)** status, then click the  **Merge** toolbar button. (Alternatively, invoke **Merge** from the file's context menu.)

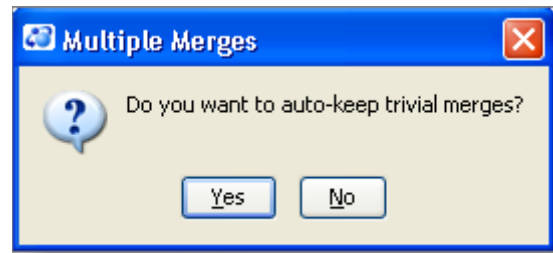


You can do this in the File Browser's details pane, either in folders view or in searches view. You can also invoke the Merge tool from the Change Palette, to merge versions in higher-level streams.

The Merge tool processes the files one-by-one. For each one, it opens a new Merge tab in the overall AccuRev GUI window. On this tab, you interactively merge these two contributor

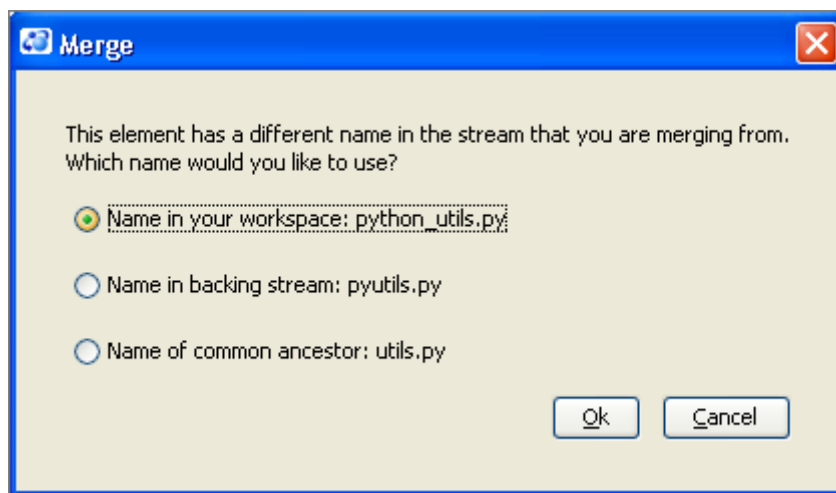
versions: (1) the file in your workspace; (2) the version in the backing stream, containing changes made by one or more of your colleagues.

If you select multiple files, you can choose to have the Merge tool merge as many of them automatically as possible. An automatic merge is possible if there are no conflicting changes between the two versions being merged (see above). For files in which there are conflicting changes, the Merge tool always opens an interactive Merge tab.



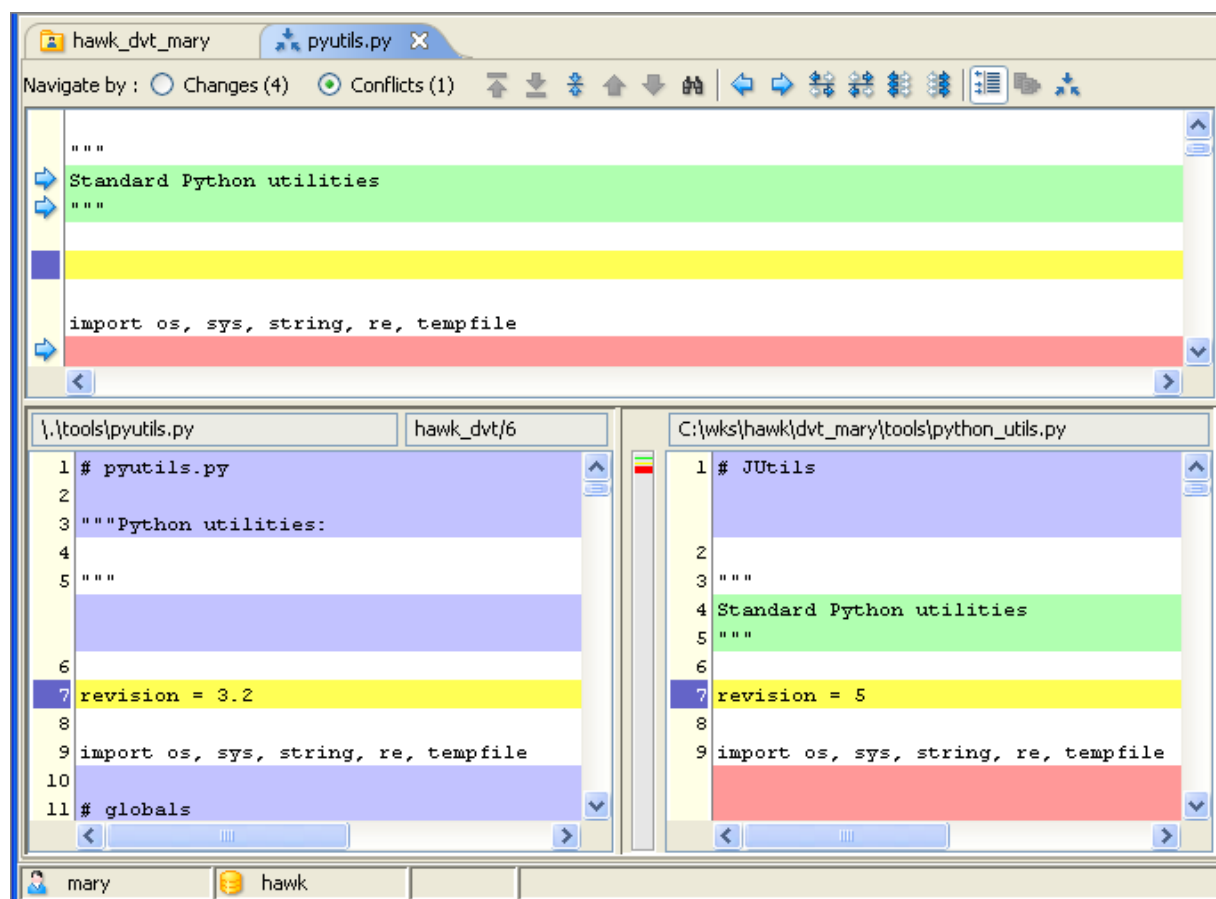
## Resolving a Namespace-Level Conflict

As outlined above, a namespace-level conflict exists if *both* contributors had their pathnames changed, and the new pathnames differ from each other. The Merge tool prompts you to resolve the conflict by choosing one of the names. You can also choose to restore the name of the common ancestor version.



## Viewing and Resolving Content-Level Conflicts

The Merge tool displays the two contributors' contents side-by-side, with the file in your workspace on the right. Above them, it displays the merged version. The bottom panes are synchronized: scrolling either one of them causes the other to scroll, too.



Like the Diff tool, the Merge tool partitions the contributors' contents, displaying unchanged sections with a white background and difference sections with colored backgrounds. The color-coding scheme is similar to the Diff tool's, but not identical. That's because the Merge tool has the more complex job of distinguishing conflicting changes from non-conflicting ones.

- For each non-conflicting change (where just one contributor differs from the ancestor version), the Diff color-coding scheme is used: the block in the backing-stream version (left side) is colored **blue**; the corresponding block in your version (right side) is colored **green** (text you've added), **red** (text you've deleted), or **blue** (text you've revised).
- For each conflict (where both contributors differ from the ancestor version), the blocks in both contributors are colored **yellow**.

The merged version is displayed with color-coding, too. Each colored block is a location where a change has been incorporated from one of the contributors into the merged version. When it starts up, the Merge tool performs as much merging as it can automatically, applying all of the non-conflicting changes to the merged version. Color indicates the origin of the change: blue if it comes from the left-hand contributor; green, red, or blue if it comes from the right-hand contributor. In addition, the change is marked with left-arrows or right-arrows, indicating which contributor provided the change.

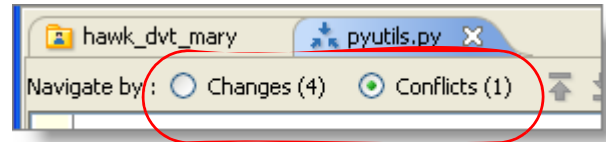
The Merge tool cannot resolve conflicts automatically, so the location of each conflict is initially indicated in the merged version by a blank yellow block.

## Navigating the Display

The Merge tool is similar to the Diff tool in its navigation facilities. The panes have scroll bars, which are synchronized with each other. A status indicator at the top of the Merge tab reports the total number of change sections (**Changes**) found in the contributors. It also tracks the current number of conflicting changes (**Conflicts**) that you have not yet resolved.

For example, the status indicators may read as shown here when the Merge tool starts up.

This means that there are a total of 5 change sections; 4 of them are non-conflicting changes, which are automatically applied to the merged version; 1 of them is a conflict, which you must resolve during the merge session.



You can browse sequentially through the change sections using the Merge toolbar buttons

⏮**Next**, ⏮**Previous**, ⏮**First**, and ⏮**Last**. Click the **Changes** radio button to have these buttons browse all the change sections; click the **Conflicts** radio button to have these buttons browse just the unresolved conflicts.


Instead of clicking the toolbar buttons, you can use these keyboard accelerators:

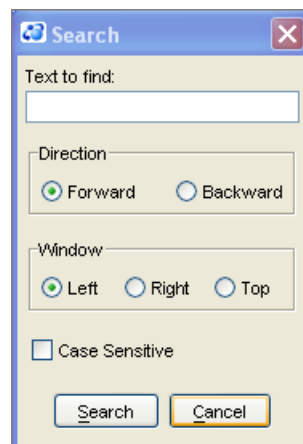
- Ctrl-F** go to first change/conflict
- Ctrl-L** go to last change/conflict
- Ctrl-P** go to previous change/conflict
- Ctrl-N** go to next change/conflict

Between the two contributor panes, there's a change map that shows the relative locations and sizes of all the change sections. The map uses color-coding to indicate the kind of change. Click on any colored area of the map to scroll all the panes directly to the corresponding change section.

Whenever you use a toolbar button or the change map to jump to a particular change section, the Merge tool remembers it as the current change. Using the scroll bar, you can make the current change scroll off screen. To make it visible again, click the **Center** button on the Merge tab's toolbar.

## Searching for Text



In addition to navigating among the change/conflict sections, you can search for any text string in any pane, using the  **Search** toolbar button — or the **Ctrl-S** keyboard accelerator.




## Resolving Conflicts

Sometimes, the Merge tool can construct a merged version without any help from you. This occurs if all the change sections are non-conflicting. The Merge tool just applies all the changes to the merged version and announces that it's done. This section describes the more interesting case: the contributors have one or more conflicts, which you must resolve.

The Merge tool automatically jumps to the first conflict. That is, it scrolls to the first yellow-highlighted change section in the contributors, and the corresponding blank yellow block in the merged version. Now, you must decide which of the changes is to be incorporated into the merged version:



- Click the  **Take their change** toolbar button to incorporate the change in the left-hand contributor.
- Click the  **Take my change** toolbar button to incorporate the change in the right-hand contributor.
- The other two buttons provide a way to incorporate *both* contributors' changes — in either order — into the merged version. Typically, you'll need to do some manual editing to “smooth out” the combined changes. See the [Manual Editing](#) section below.

When you click the button, the selected change appears in the merged version and the highlight changes from yellow (unresolved) to blue (resolved). Also, the Conflicts counter in the Merge toolbar is decremented.



You've resolved one conflict; all you need to do is resolve the rest of them in the same way. Use the  **Next** button to scroll down to the next conflict. It may take more than one click if sequential browsing is in Changes mode rather than in Conflicts mode. Again, use one of the “take changes” buttons to select the text from the left contributor or the right contributor (or both of them).

You don't have to resolve the conflicts in order. You can jump around as much as you want among the change sections, or scroll through unchanged sections to look up information the affects your

merge decisions. The Conflicts counter always shows how much more work you need to do to complete the merge.

You can also change your merge decisions as much as you want. For example, you can revisit a particular change section where you had selected  **Take their change** and click the  **Take my change** button. This switches the text incorporated into the merged version.

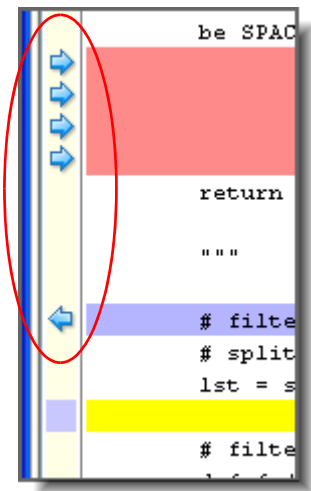
### The ‘Revert All’ and ‘Use Only’ Buttons

There are two additional toolbar buttons for resolving conflicts. The  **Revert all of my changes** button makes the merged version the same as the “from” version (typically, the version in the backing stream). The  **Use only my changes** button makes the merged version the same as the “to” version (the version in your workspace).

Here’s a good way to think of this. The merged version has arrow annotations that indicate how each conflict has been resolved by you, and how each non-conflicting change has been resolved automatically by the Merge tool. “Revert all of my changes” makes all these arrows point to the left; “Use only my changes” makes all these arrows point to the right.

It’s important to note that these operations can undo the Merge tool’s automatic resolution of one or more non-conflicting changes.

**“Revert all of my changes” makes all these arrows point left**  
**“Use only my changes” makes all these arrows point right**



### Manual Editing

At any time during a merge session, you can manually edit the contents of the merged version. Just click anywhere in the pane containing the merged version, and type. The **Delete** and **Backspace** keys work as expected. Using context (right-click) menus, you can **Copy** and **Paste** sections of text that you’ve highlighted with the mouse. You can also use the common keyboard shortcuts: **Ctrl-C** or **Ctrl-Ins** to copy, **Ctrl-V** or **Shift-Ins** to paste.


Notes:

If you make changes manually within a conflict section that you’ve resolved with a **Take ... change** button, the manual changes will be overwritten if you click any of the **Take ... change** buttons again.

Selection of text sections using the keyboard is not currently supported.

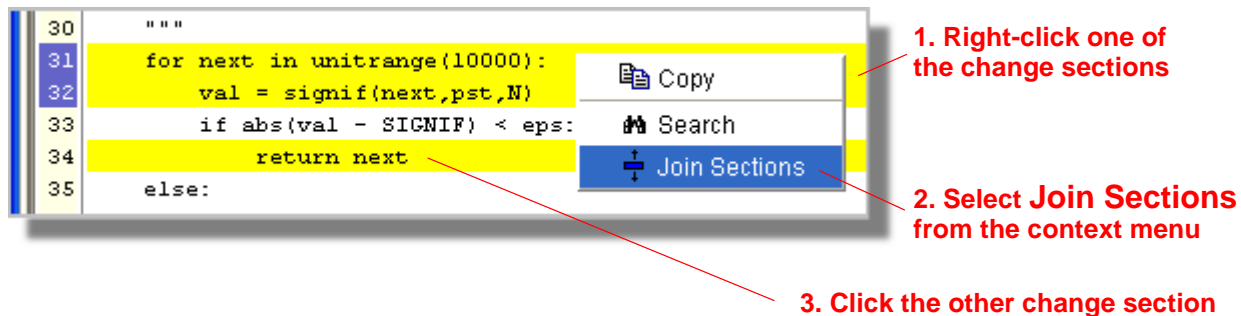


## Searching for Text

As with the Diff tool, you can search for any text string in either contributor pane, using the  **Search** toolbar button. You can also invoke this command from the context menu that appears when you right-click anywhere in one of the contributor panes.

## Joining Change Sections

Sometimes, there is a block of lines that you consider to be a single change section, but that the Merge tool decides are two discrete sections. You can tell the Merge tool to combine the two change sections:




## Finishing a Merge Session

When you resolve the last remaining conflict by clicking one of the “take changes” buttons, the Conflicts counter goes to zero and the Merge tool displays this message window:

Clicking **Keep & Exit** ends the merge session immediately, closing the Merge tab. AccuRev overwrites the file in your workspace with the merged version. (Thus, you can think of the Merge tool as a fancy text-editor — it modifies the contents of a file in your workspace.) Then, it immediately preserves this modified file by **keeping** a new version in your workspace stream. The **overlap** status of the file is removed, so that you can now **promote** the new version to the backing stream.




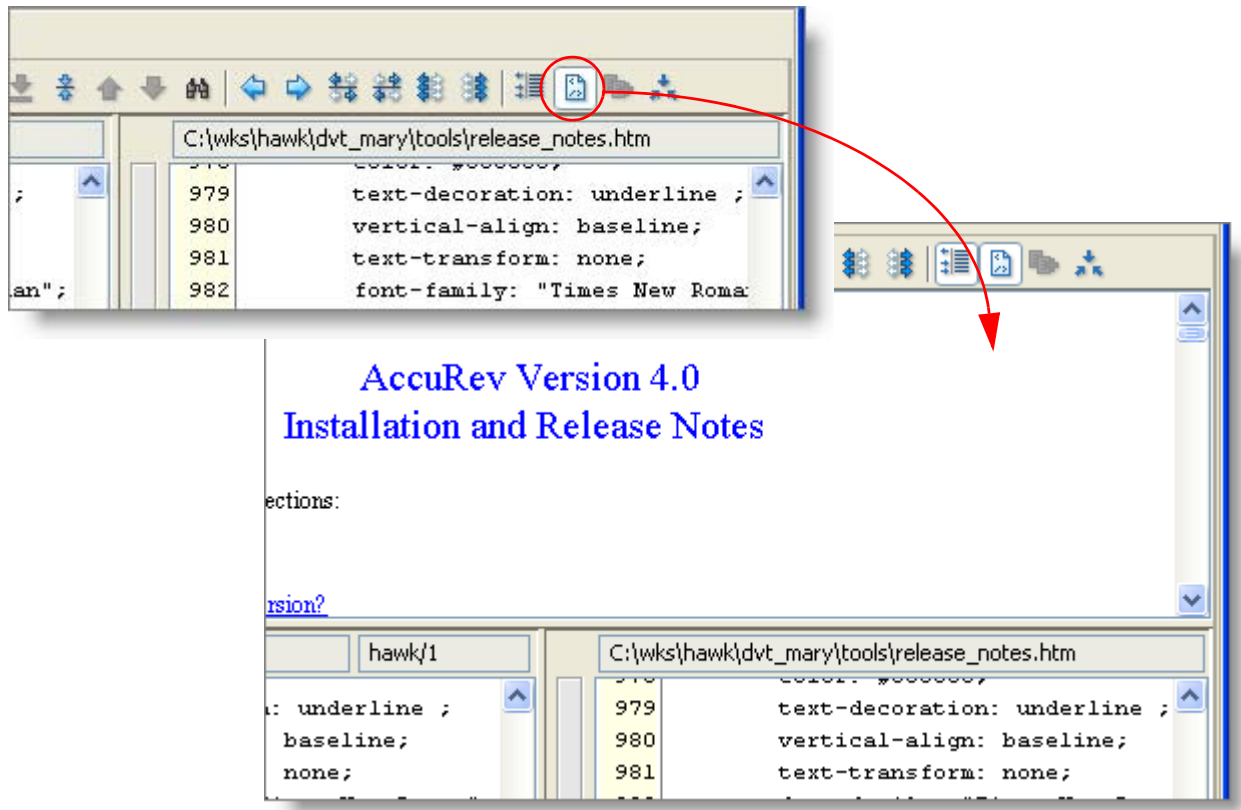
Clicking **Return** continues the merge session. You can review your work, change some of your merge decisions, and perform manual edits. At any time, you can click the  **Keep** button in the Merge toolbar to complete the merge process.

You can also cancel the entire merge during a review pass by closing the Merge tab without keeping: right-click on the tab title, then confirm.

## Merging HTML Files

AccuRev's merge algorithm treats HTML-format files just like all other text files: merging takes place line-by-line; no attempt is made to parse the files' HTML data structures in determining difference sections.

To help you determine whether merging is producing a valid (and desirable) result, though, an HTML viewer is built into the Merge tool. At any time, you can click the  **Show HTML Result** toolbar button to render the current contents of the merged version. In the upper pane, the merged HTML-format text is replaced by a rendering of the text.



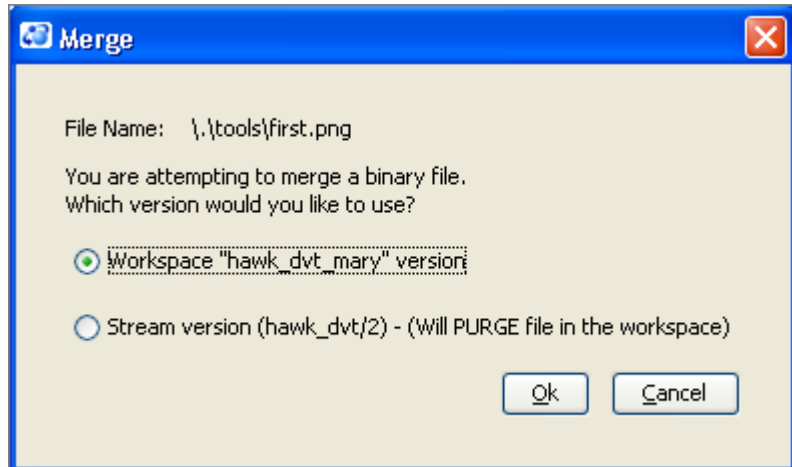
This is a toggle button; click it again when you're finished viewing the rendered HTML code and you want to return to performing the merge.

## Merging Binary Files

No generally accepted algorithm exists for merging the contents of binary-format files. But it is quite possible for a binary-format element to get into an overlap state in a concurrent development environment. For example: team members Justine and Mary each revise the corporate logo, **first.png**, using an image editor. Justine keeps and promotes the file to the backing stream, creating an overlap situation for that file in Mary's workspace.

To resolve the overlap, Mary invokes the **Merge** command, just as she would for a text file. The Merge tool, seeing that **first.png** is in binary format, offers her the only two possible choices:

- Mary can select **Workspace version** to retain her changes to the file. A new version is created in Mary's workspace, recording a merge from the backing-stream version. This allows Mary to promote her version to the backing stream.
- If she selects **Stream version**, Mary's changes are purged from her workspace. The workspace reverts to using the version of the file that was current at the previous update. The file's status becomes stale, reflecting the fact that Justine has created a new version in the backing stream since that update. If she wishes, Mary can now bring Justine's version into her workspace with the **Update** command.



## Performing a Patch Instead of a Merge

Note: operationally, performing a patch is exactly the same as performing a merge. Using the Merge tool, you combine text from some other source with the contents of the file in your workspace. This section explains the difference between the patch and merge operations from an SCM perspective.

When you merge version V of a file into your workspace version, you are essentially saying, “combine my file with version V, taking into account all the changes in version V, back to the common ancestor”. By contrast, when you patch version V into your workspace version, you are saying, “combine my file with version V, taking into account *only the recent changes* to version V”.

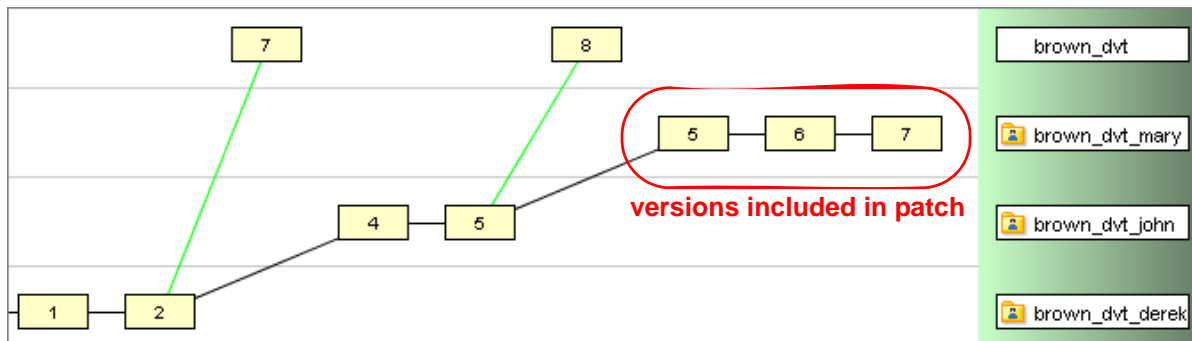
In AccuRev 3.5.5, the meaning of “only the recent changes” was modified to make the version-patching facility more effective and less labor-intensive. In previous releases, it meant “just the changes between version V and its immediate predecessor”. For example, suppose user Mary recently created versions 4, 5, 6, and 7 of a file in her workspace, **dvt\_mary**. When patching version **dvt\_mary/7** into your workspace version, AccuRev previously only incorporated the changes between versions **dvt\_mary/6** and **dvt\_mary/7**. The drawback of this simple algorithm is clear: you probably wanted to incorporate *all* of Mary's recent changes — the modifications in versions 4, 5, 6, and 7. In previous releases, you would have had to perform four separate **Patch** commands to achieve this result.

AccuRev now implements a more reasonable definition of “only the recent changes in Version V”. It scans backward through the file's ancestry, starting at version V, stopping when it encounters a version that was originally created in another workspace, or a version that was

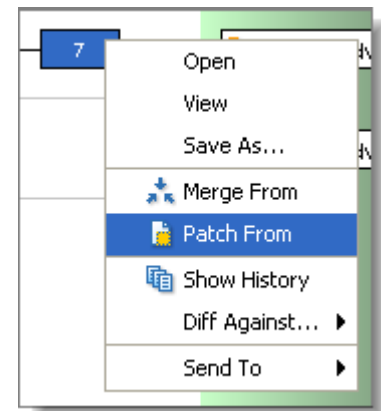
promoted to another stream. This older version, termed the basis version, is judged to precede (and not belong to) the set of “recent changes” in version V.

## Patch Example 1

Before Mary started her recent work, she updated her workspace. This brought in a version of the file originally created by another user — say, version **brown\_dvt\_john/5**. Then she proceeded to create versions 5 through 7 in workspace **brown\_dvt\_mary**.

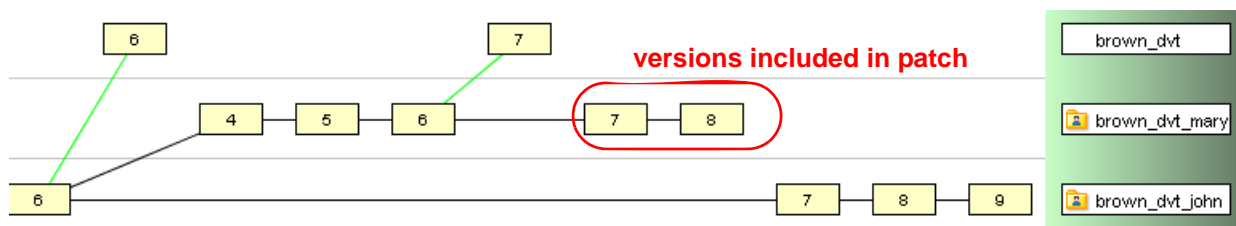


Derek decides to patch version **brown\_dvt\_mary/7** into his workspace. He invokes the **Patch From** command from the context menu of this version. AccuRev searches backward through the element’s ancestry, and includes the set of changes recently made in Mary’s workspace: this set includes **brown\_dvt\_mary/5** through **brown\_dvt\_mary/7**; the patch doesn’t include the two versions created in workspace **brown\_dvt\_john**.



## Patch Example 2

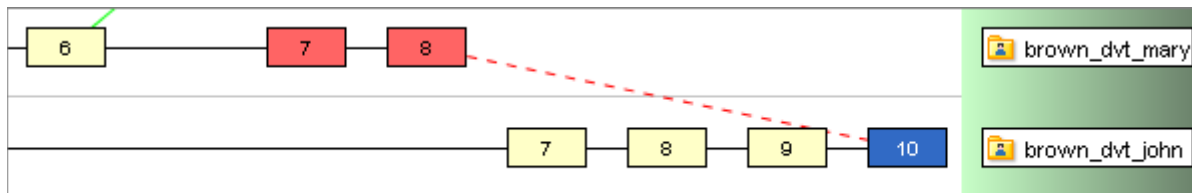
Suppose Mary started by bringing version **brown\_dvt\_john/6** into her workspace with an update. Then she created versions **brown\_dvt\_mary/4** through **brown\_dvt\_mary/6**, promoted her work to the backing stream, and created two more versions: **brown\_dvt\_mary/7** and **brown\_dvt\_mary/8**.



When John patches version **brown\_dvt\_mary/8** into his workspace, AccuRev decides that only the versions since the promotion — versions 7 and 8 — contain “recent changes”. The idea is that a promotion typically marks the end of a programming task, not an intermediate checkpoint.

## Indicating a Patch and its Set of Versions

The Version Browser uses a dashed red line to indicate a version created by a **Patch** command. Selecting the version causes the set of versions included in the patch to be highlighted in red.



The basis version is recorded in the **keep** transaction that records a **Patch** command in the repository:

```
transaction 206; keep; 2005/12/29 15:57:57 ; user: john
# patched
\\.dir00\sub02\file03.txt 5/10 (5/10)
ancestor: (5/9) patched from: (4/8)-(4/6)
```

In this command, version 4/8 (**brown\_dvt\_mary/8**) was patched into version 5/9 (**brown\_dvt\_john/9**), and the result was kept as version 5/10 (**brown\_dvt\_john/10**). The backward search through the file’s ancestry identified 4/6 (**brown\_dvt\_mary/6**) as the basis version.


# The History Browser

The AccuRev GUI's History Browser is a tool for viewing some or all of the transactions associated with a particular element, stream, or depot.

## Invoking the History Browser

There are several ways to launch the History Browser:

- **Transactions Involving an Individual Element.** You can have the History Browser display the transactions in which one particular element was involved. (Other elements may have been involved in these transactions, too.) Elements are listed in various places in the GUI:
  - In the details pane of a File Browser tab
  - In the Default Group Contents subwindow of a StreamBrowser
  - In an issue record's change package (Changes subtab)

In any of these contexts, you can select an element and click the  **Show History** toolbar button. Alternatively, right-click an element and select **History > Show History** from the context menu.


- **Transactions Involving a Particular Stream.** You can have the History Browser display transactions that affected a particular stream. For a workspace stream, this principally includes the **keep** transactions that create real versions in the stream. It can also include **co**, **move**, **defunct**, and **undefunct** transactions. For a dynamic stream, this includes **promote** transactions *to* the stream, but not promotions *from* the stream.

In the StreamBrowser, right-click a stream and select **Show History** from the context menu.

- **Active Transactions for a Particular Stream.** Every stream has a default group, consisting of the elements that are active in that stream. In the StreamBrowser, a stream's context menu includes the command **Show Active Transactions**; it opens a History Browser tab and loads the set of transactions (**keep**, **promote**, etc.) in which the stream's active versions were created.
- **Transactions Involving a Particular Depot.** You can view the transactions for an entire depot — all the elements, in all the streams. Use the **Admin > Depots** command to list all of the repository's depots. Then, select a depot and choose **Show History** from its context menu, or from the **Actions** group in the main menu.

There are also contexts in which you can invoke the History Browser to view a single transaction, or a specific list of them, rather than the collections described above:

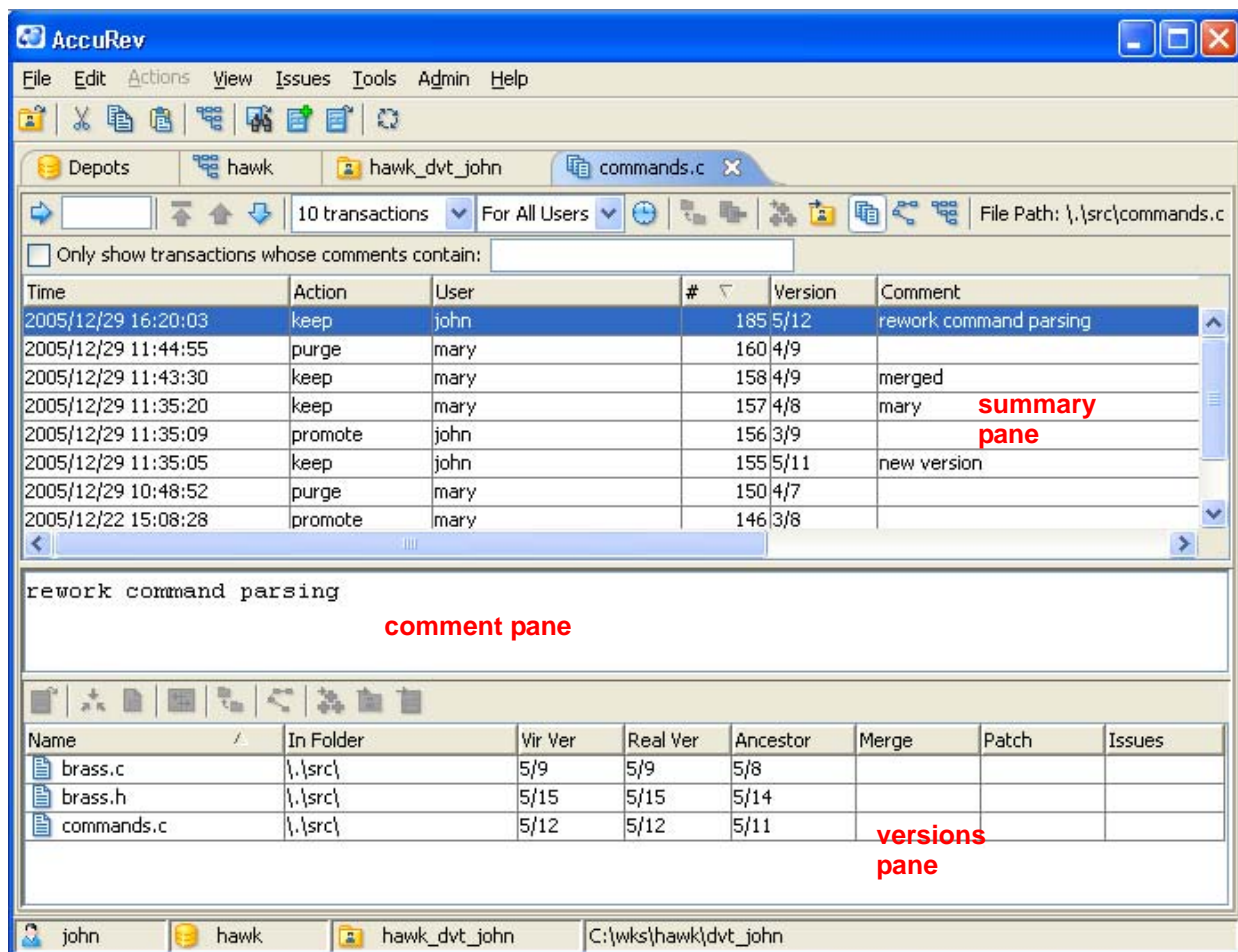
- In the Version Browser, selecting **Show History** from any version's context menu displays the transaction in which that particular version was created.

- If you've used the transaction-level integration between configuration management and issue management, an issue record's **affectedFiles** field contains a list of transaction numbers.  
Clicking the  **Show History** button next to this field displays just those listed transactions.

## The History Browser Display

The History Browser appears in a separate tab of the AccuRev GUI's multiple-tab display. The tab is divided into three panes:

- The **summary** pane displays a group of transactions, one per line. This pane displays overall information: transaction number, timestamp, transaction type, etc.
- The **comment** pane shows the comment string, if any, that was specified for the currently-selected transaction.
- The **versions** pane shows all the versions that were involved in the currently-selected transaction. It also indicates which change packages, if any, those versions belong to (Issues column).



AccuRev

File Edit Actions View Issues Tools Admin Help

Depots hawk hawk\_dvt\_john commands.c

10 transactions For All Users File Path: \.\src\commands.c

☐ Only show transactions whose comments contain:

Time	Action	User	#	Version	Comment
2005/12/29 16:20:03	keep	john	185	5/12	rework command parsing
2005/12/29 11:44:55	purge	mary	160	4/9	
2005/12/29 11:43:30	keep	mary	158	4/9	merged
2005/12/29 11:35:20	keep	mary	157	4/8	mary
2005/12/29 11:35:09	promote	john	156	3/9	
2005/12/29 11:35:05	keep	john	155	5/11	new version
2005/12/29 10:48:52	purge	mary	150	4/7	
2005/12/22 15:08:28	promote	mary	146	3/8	

rework command parsing

Name	In Folder	Vir Ver	Real Ver	Ancestor	Merge	Patch	Issues
brass.c	\.\src\	5/9	5/9	5/8			
brass.h	\.\src\	5/15	5/15	5/14			
commands.c	\.\src\	5/12	5/12	5/11			

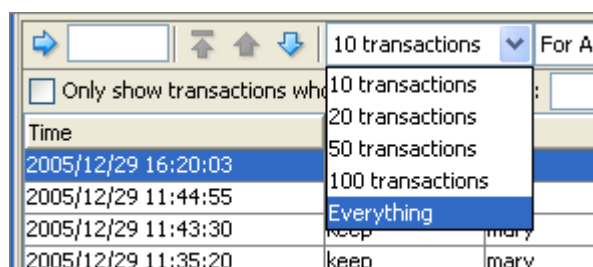


## The Summary Pane

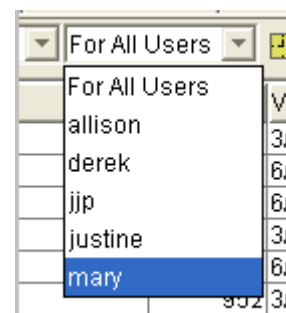
The summary pane displays a table containing a set of transactions, one per row. You can manipulate the table in the standard ways: adjusting column widths, rearranging columns, sorting the rows on one or more columns.

There's potentially a large number of transactions to display in this table — e.g. when displaying the history of an entire depot. Retrieving all the transactions at once from the depot's database can be time-consuming. The History Browser handles this situation by initially retrieving a small number of transactions, then letting you control the retrieval of additional transactions with the toolbar located just above this table.

The table initially contains the 10 most recent relevant transactions. To change this count, use the Transaction Count list box. (Each time you change the count, the browser returns to displaying the most recent transactions.) Selecting **Everything** loads all the relevant transactions into the summary pane.

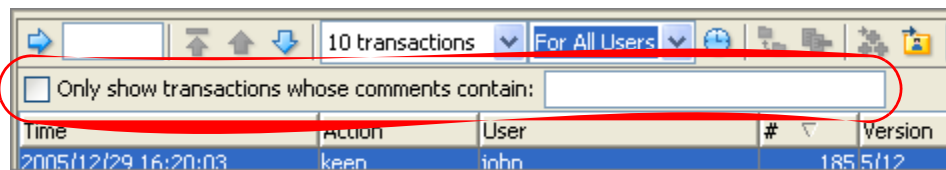


By default, the browser displays transactions created by all users. To restrict the display to one user's transactions, use the User Filter list box.




To filter the display based on transactions' comment fields, type a search string in the Comment

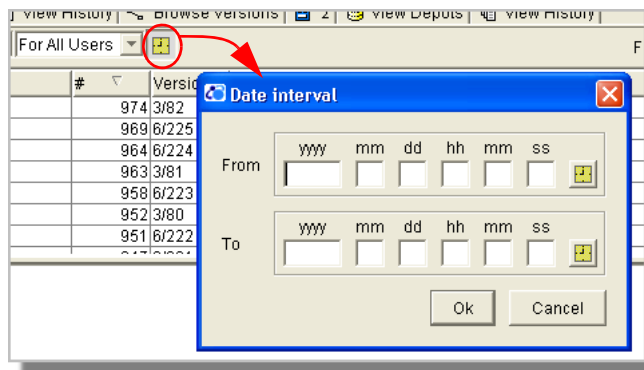
Filter input field, then click the checkbox to enable the filtering.








Use the  **Set Date Interval** toolbar button to restrict the display to transactions that occurred in a specified interval.

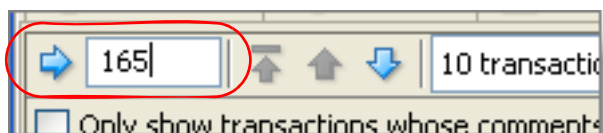
This changes the setting in the Transaction Count list box to **Custom**. To cancel the date interval restriction, change this list box setting back to **10 Transactions** (or something else).



Once a certain set of transactions has been loaded into the summary pane, you can browse through those transactions. Click any transaction to select it; the Comment and Versions panes are updated with the transaction's details. You can use the usual navigation keys to change the selected transaction: up-arrow and down-arrow, **PgUp** and **PgDN**, **Ctrl-PgUp** and **Ctrl-PgDn**.

Several controls on the toolbar enable you to retrieve additional transactions, which have not yet been loaded into the summary pane:

- The  button unloads the current set of transactions, then loads the next older set of transactions.
- The  button unloads the current set of transactions, then loads the next newer set of transactions. This button is enabled only when you're viewing the history of an entire depot.
- The  button unloads the current set of transactions, then loads the first (newest) set of transactions.
- Entering a transaction number in the Goto field unloads the current set of transactions, then loads the set beginning with the specified number. (If that particular transaction is not relevant — e.g. it did not involve the element whose history you're viewing — the set begins with the next older relevant transaction.)



For all of these controls, the Transaction Count, User Filter, and Date Interval settings determine exactly how many transactions (and which ones) are retrieved from the depot's database and loaded into the summary pane.

Each row of the summary pane's table displays the following information about an individual transaction:

### Time

A timestamp, indicating when the transaction took place.

### Action

The kind of transaction: **keep**, **promote**, etc.

## User

The principal-name of the user who initiated the transaction.

## # (transaction number)

The unique number (within this depot) of the transaction.

## Version

(only for transactions involving an individual element) The real version or virtual version of the element that was created in this transaction. This column doesn't appear in displays of a stream's history or an entire depot's history; the version(s) created by the transaction appear in the Versions pane.

## Comment

The first line of the user-supplied comment for this transaction. If the comment extends to additional lines, an ellipsis ("dot dot dot") appears here. For the full text of the comment, look in the Comment pane.

## Operations on Transactions in the Summary Pane

You can perform numerous operations on a selected transaction in the summary pane, using its context menu.

Time	Action	User	# ▾	Version
2005/12/29 16:20:03	keep	john	185	5/12
2005/12/29 11:44:55	purge	Promote	160	4/9
2005/12/29 11:43:30	keep	Revert	158	4/9
2005/12/29 11:35:20	keep	Diff Against...		Other Version
2005/12/29 11:35:09	promote	Send To		File in the Workspace
2005/12/29 11:35:05	keep			
2005/12/29 10:48:52	purge	mary	150	4/7

## Promote

*This command is not implemented in the current release.*

(from Show Active Transactions only) Promote all of the transaction's versions to the parent stream of the stream/workspace from which you invoked the History Browser.

## Revert

*This command is not implemented in the current release.*

(from dynamic stream history only) For the selected **Promote** transaction, performs the equivalent of an "undo" in a particular child workspace of the dynamic stream. AccuRev prompts you to specify which workspace. Older versions of the transaction's elements will be activated in that workspace. For details, see the **revert** reference page in the *AccuRev User's Guide (CLI)*.

## Diff Against Other Version

(from text-file element history only) Compare the version in the selected transaction with the version in another transaction. AccuRev prompts you to select another transaction from the summary pane.

## Diff Against File in the Workspace

(from text-file element history only, when in the context of a particular workspace) Compare the version in the selected transaction with the file in the workspace from which the History Browser was invoked. This workspace is listed at the bottom on the AccuRev GUI window. This enables you to invoke a comparison with a file that you have changed in your workspace, but have not yet preserved in the repository with a **Keep** command.

This command is disabled if you invoked the History Browser from a non-workspace context.

## Send to Change Palette

(from dynamic stream history or Show Active Transactions only) Record each of the transaction's versions in the Change Palette, for promotion to another stream.

## Send to Workspace

Activate each of the transaction's versions in a particular workspace. AccuRev prompts you to specify one of your workspaces.

## The Comment Pane

The Comment pane displays the full text of the user-supplied comment for this transaction.

## The Versions Pane

The Versions pane displays a table of all the versions, if any, created by the currently-selected (highlighted) transaction. Both the real version and virtual version IDs are listed. (For transactions in a workspace, these IDs are the same; for transactions in a dynamic stream, they differ.)

For transactions in a workspace, the Ancestor column indicates the version that the newly created version was derived from. For a **Keep** transaction that was performed by a **Merge** command, the Merge column indicates the "from" version in the merge. Similarly, the Patch column indicates the "from" version for a **Keep** transaction that was performed by a **Patch** command.

The Issues column indicates the change package(s) to which each version belongs.

You can manipulate this table in the standard ways: adjusting column widths, rearranging columns, sorting the rows on one or more columns.

## Operations on Versions in the Versions Pane

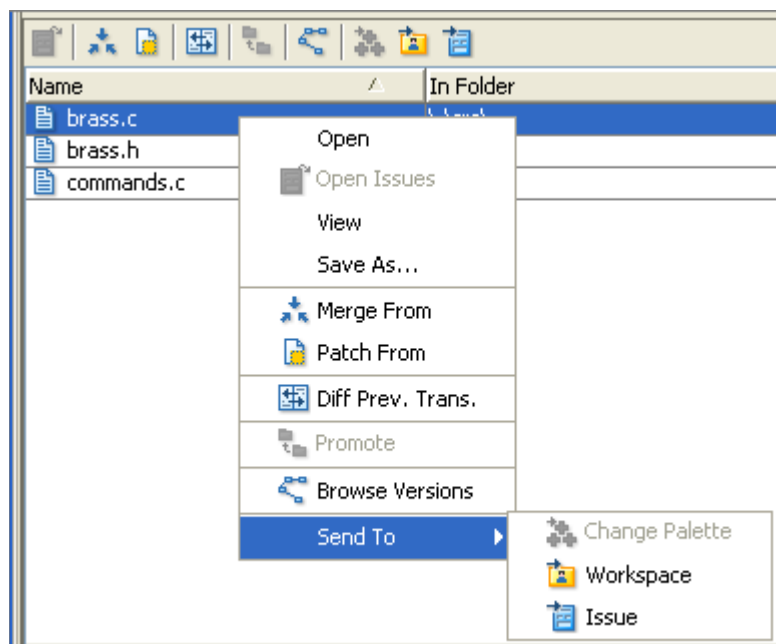
You can perform several operations on a selected version in the Versions pane, using its context menu or the versions pane toolbar.

### Open

Run the appropriate command on the file, according to its file type

### Open Issues

Displays the change package(s) — that is, the issue record(s) — to which the version belongs. For a version that belongs to a single change package, an edit form opens on that one issue record. For a version that belongs to multiple change packages, an Issues tab opens, listing all the issue records.



### View

Open a text editor on a temporary copy of the currently selected file (text files only).

### Save As

Copy the currently selected file to another filename.

### Merge From

(History Browser invoked from a File Browser only) Merge the selected version into the version in the workspace from which the History Browser was invoked.

### Patch From

(History Browser invoked from a File Browser only) Patch the selected version into the version in the workspace from which the History Browser was invoked.

### Diff Against Previous Transaction

Compare the selected version with the version that was in the current workspace or stream just prior to the selected transaction.

### Promote

*This command is not implemented in the current release.*

Promote the selected version to the parent of the stream/workspace from which the History Browser was invoked.

## Browse Versions

Open a Version Browser on the element whose version you selected.

## Send to Change Palette

(dynamic stream only): Load the selected versions into the Change Palette, so that they can be promoted to other streams.

## Send to Workspace

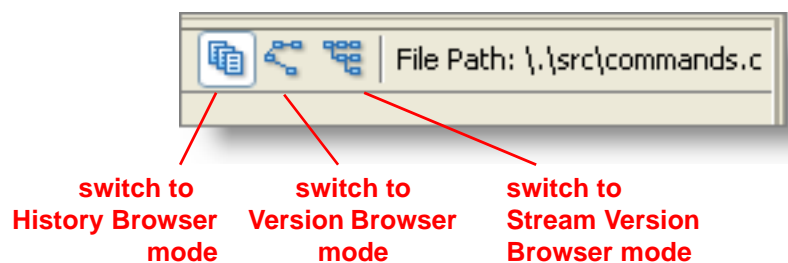
(History Browser invoked from a File Browser only) Activate the selected element in the workspace from which the History Browser was invoked.

## Send to Issue

Record the selected version(s) in the change package section (Changes tab) of one or more issue records. The default query of the issues database is executed, and you are prompted to choose one or more of the records selected by the query. You can also create a new issue record, to which the selected version(s) will be sent.

## Quick Switch: History / Version / Stream Version Browsers

In the toolbar of all History Browser, Version Browser, and Stream Version Browser tabs, there are three buttons in the upper right corner. These buttons enable you to perform a “quick switch”, for example to change a Version Browser tab into a History Browser tab.




You can think of the three browsers as the several display modes of a single “Versions” tool. The multiple modes provide different ways to view some or all of the versions of an element.

# The Version Browser: Ancestry Tracking

AccuRev maintains complete ancestry information for each element. The Version Browser displays some or all of an element's versions, using color-coded lines to indicate the way in which each version was created.

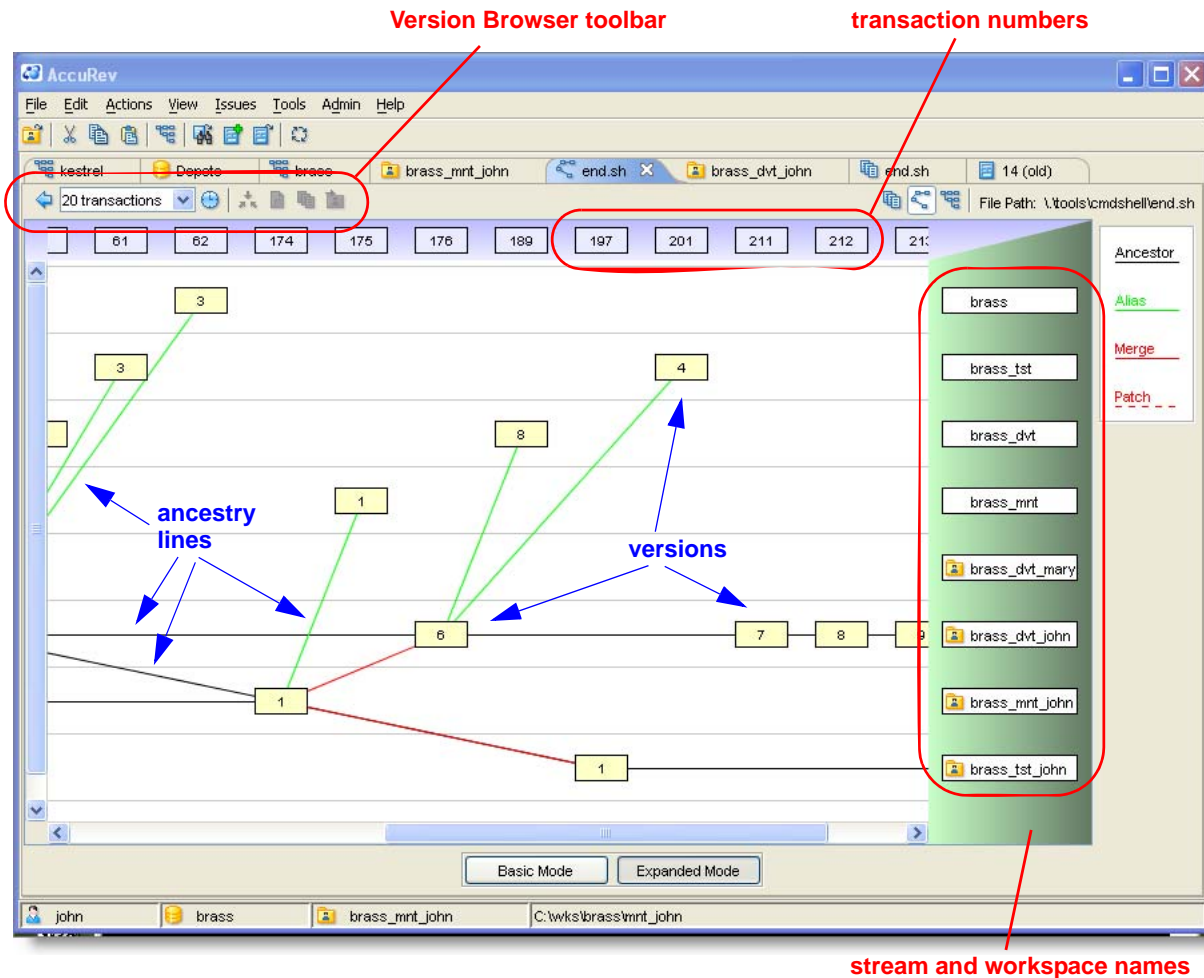
## Invoking the Version Browser

The names of elements appear in many contexts in the AccuRev GUI. These include the details pane of the File Browser, the versions pane of the History Browser, and the Changes tab (change package) of a AccuWork issue record. In any of these contexts, you can select one element and

invoke the  **Browse Versions** command. (In some cases, this command is a subchoice, under a **History** menu choice.)

## The Version Browser Display

The Version Browser display contains a yellow box for each version of the specified element. These boxes are arranged in rows, according to the workspace or stream in which the version was created. The name of the workspace or stream appears at the right edge of the display. Above each version box, at the top of the display, is a blue rectangle that represents the transaction in which the version was created. Color-coded ancestry lines connect the versions, indicating how the later version was derived from the earlier version.



The kinds of ancestry lines are:

- **direct ancestor** (black) — A user started with the earlier version, modified (or overwrote) it, and then performed a **Keep**, creating the later version. (There are a few other commands, including **Rename**, that might have created the later version. See below for details.)
- **alias** (green) — A user promoted the earlier version to create the later version.
- **merge** (red) — A user executed the **Merge** command to create the later version. There are always two earlier versions: the black line indicates the direct ancestor (see above) of the later version; the red line indicates the version that was merged with the direct ancestor.
- **patch** (dashed red) — Similar to **merge** above; instead of the **Merge** command, the **Patch** command was used to create the later version.

## User Preferences

The **Tools > Preferences** command opens a dialog box that includes a **Version Browser** tab. On this tab, you can control these parameters:

## Initial display mode

Specifies the mode that a new Version Browser tab begins in. **Basic** mode displays “important” versions only, for example omitting intermediate versions in workspace streams. **Expanded** mode displays all versions. Buttons at the bottom of the Version Browser tab switch between the two display modes.

## Initial transaction count

How many of the most recent transactions (and equivalently, versions) a new Version Browser should display. Controls in the Version Browser toolbar change this count.

See [User Preferences](#) on page 6.

## Ancestry Relationships

The following sections discuss the four kinds of ancestry in more detail, along with the important concept of closest common ancestor.

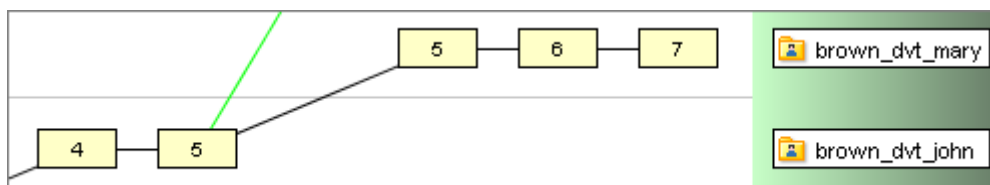
### Direct Ancestor — Modification of an Existing Version

Probably the most common AccuRev operation is starting with an existing version, making changes, and then executing the **Keep** command to save the changes in a new version. The existing version might have been created by you — for example, with a previous **Keep** command. Or it might have been created by someone else: that user **Promote**’d the version, then you brought it into your workspace with an **Update**.

The version in your workspace created by the **Keep** command is termed a real version, because it represents a change to the element. Other commands can create real versions in your workspace, too: **Rename**, **Defunct**, **Undefunct**.

The Version Browser uses a black line to connect the existing version (termed the direct ancestor or predecessor) with the new version.

In this figure:



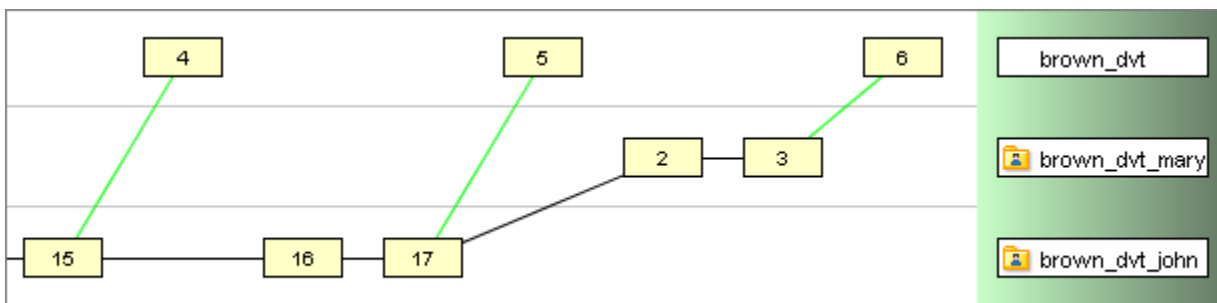
- Version 4 in the **brown\_dvt\_john** workspace stream was edited and saved with **Keep**, to create version 5 in the same stream.
- Version 5 in the **brown\_dvt\_john** workspace stream was edited and saved with **Keep**, to create version 5 in the **mary** stream.
- Version 5 in the **brown\_dvt\_mary** workspace stream was edited and saved with **Keep**, to create version 6 in the same stream; and version 6 was edited and saved with **Keep**, to create version 7.



- Version 1 in the **brs\_wk\_mary** stream was edited and saved with **Keep**, to create version 2 in the same stream.
- Version 2 in the **brs\_wk\_mary** stream was edited and saved with **Keep**, to create version 3 in the **brs\_wk\_jjp** stream.

## Alias — Virtual Version Ancestry

Workspaces contain real versions, which represent changes to elements. By contrast, all versions in dynamic streams are virtual versions, created with the **Promote** command. Each virtual version is an alias for — that is, a pointer to — some real version in a user's workspace. The Version Browser uses a green line to connect a virtual version in a dynamic stream to the corresponding real version in a workspace.



Note: there's one exception to this scheme. The **Anchor** command (and its CLI analog, **co**) create a virtual version in a workspace. It's a virtual version because it doesn't represent a change to the element, but merely the restoration of an existing version to the workspace.

In the figure above:

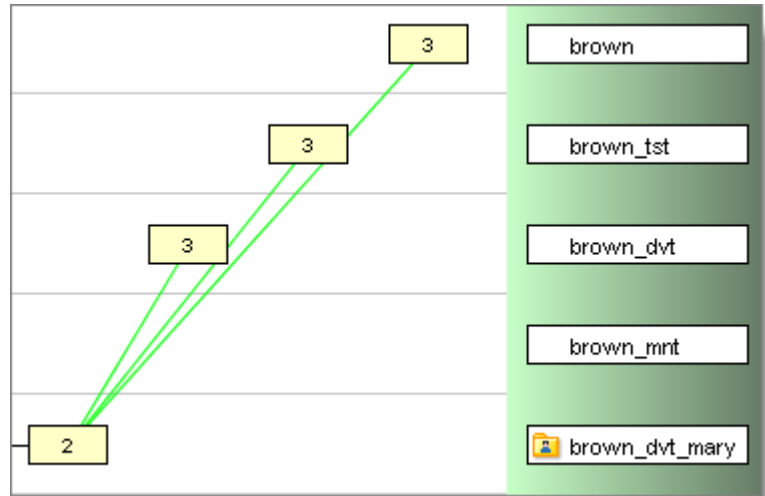
- version 4 in stream **brown\_dvt** is an alias for (was promoted from) version 15 in workspace **brown\_dvt\_john**
- version 5 in stream **brown\_dvt** is an alias for version 17 in workspace **brown\_dvt\_john**
- version 6 in stream **brown\_dvt** is an alias for version 3 in workspace **brown\_dvt\_mary**

In a depot with a deep stream hierarchy, it's common to successively promote a particular version to the parent stream, then to the grandparent stream, then to the great-grandparent stream, etc. All the versions created by this series of **Promote**'s are aliases for the same real version. The Version Browser shows how all the virtual versions relate back to the original real version.

In the figure at right, the versions in streams **brown\_dvt**,

**brown\_tst**, and **brown** are all

aliases for the real version 2 in workspace stream **brown\_dvt\_mary**. (The display does not indicate the fact that the version was promoted from **brown\_dvt** to **brown\_tst**, and from **brown\_tst** to **brown**.)



## Merge — Merging of Two Versions

A standard **merge** operation combines the contents of these two versions of a file:

- The most recently kept version in your workspace stream. This version contains the changes that you have made to the file in your workspace.

Note: it's possible — but not a best practice — to perform a merge on a file with **(modified)** status. Since you haven't preserved the recent changes with **Keep**, AccuRev will have no way to restore the file to its state just before the merge. This can be painful if you change your mind or make an error during the merge process.

- The most recent version in the backing stream.

The result file of the merge operation is kept as a new version in the workspace stream. (You can think of merging as a fancy text-editing operation; as with any edit to a file, you preserve the results with **Keep**.) This new, merged version has two ancestors: the two versions listed above.

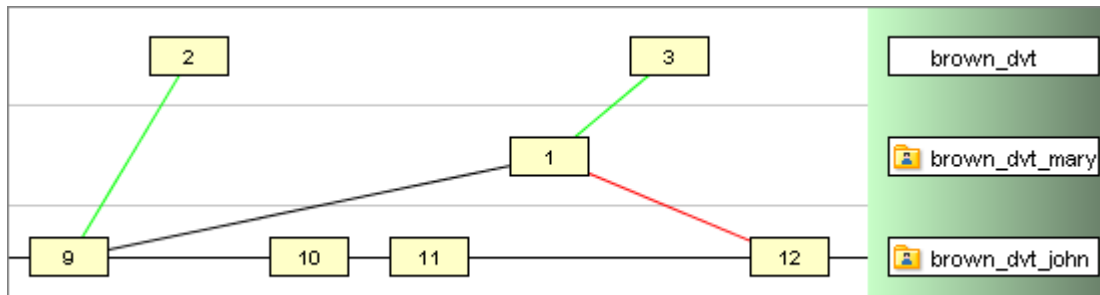
This is all simple enough. There's a twist, though, which shows up in the Version Browser display: AccuRev always records real versions, not virtual versions, as the two ancestors of a new, merged version. Thus, the ancestors in the standard merge scenario described above are:

- The most recently kept version in your workspace stream.
- The version in some other workspace stream that was promoted to the backing stream, causing the overlap that necessitated the merge.

The screen shot below shows a merge from the backing stream **brown\_dvt** to the workspace stream **brown\_dvt\_john**. The new, merged version is **brown\_dvt\_john/12**. Its ancestors are:

- Real version **brown\_dvt\_john/11**

- Real version **brown\_dvt\_mary/1**, which was promoted to become virtual version **brown\_dvt/3** in the backing stream.



A solid red line shows the merging of data from one workspace, **brown\_dvt\_mary**, to a different workspace, **brown\_dvt\_john**. The black line (“direct ancestor”) between versions 11 and 12 in the **brown\_dvt\_john** workspace reflects the viewpoint that merging is just a fancy text-editing operation, automating the creation of the next version of a file in a workspace.

### Closest Common Ancestor

It’s instructive to follow all the black and solid-red lines in an element’s Version Browser display. This traces the entire ancestry of real versions of an element. In particular, you can use the real-version ancestry to determine the closest common ancestor of any two versions. This is the most recent version upon which the two versions are both based, by some combination of ancestor and merge connections. (When considering a virtual version in a closest-common-ancestor analysis, first follow the green line back to the corresponding real version.)

You can also use the CLI command **accurev anc -c** to find the closest common ancestor of two versions.

The **merge** command determines the closest common ancestor of the two versions to be merged, and uses this version to perform a 3-way merge. This merge algorithm evaluates each difference between the two versions as a *change* — in one or both versions — from the closest common ancestor.

### Patch — Selective Merging of Two Versions

A patch operation is similar to a merge operation. In both, text from another version (the “from” version) is incorporated into your workspace’s version. Here’s the difference:

- A merge operation considers the entire contents of the “from” version.
- A patch operation considers only the parts of the “from” version that are “recent changes”.

For a discussion of the “recent changes” concept, see [Viewing and Resolving Content-Level Conflicts](#) on page 125.

Note: AccuRev tracks patch ancestry separately from merge ancestry. In determining the closest common ancestor of two versions for a merge operation, AccuRev takes into account previous merge operations (solid red), but not previous patch operations (dashed red).

The **patchlist** command compares two versions of a text element; it expresses the difference as a list of versions that need to be patched into the second version, in order to incorporate all the changes that appear in the first version.

## Operations on Versions

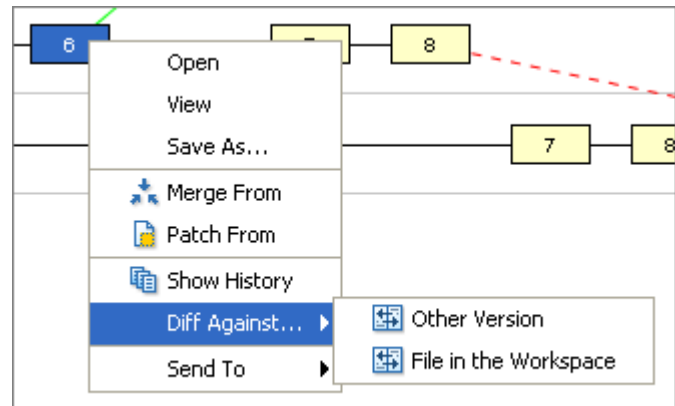
You can perform several operations on a selected version, using its context menu or the Version Browser toolbar.

### Open

Run the appropriate command on the file, according to its file type

### View

Open a text editor on a temporary copy of the currently selected file (text files only).



### Save As

Copy the currently selected file to another filename.

### Merge From

Merge the selected version into the version in the workspace from which the Version Browser was invoked.

### Patch From

Patch the selected version into the version in the workspace from which the Version Browser was invoked.

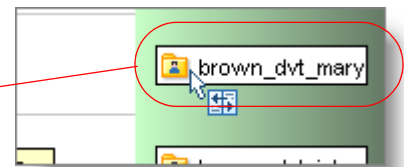
### Show History

Open a History Browser tab, showing the transaction that created the selected version.

### Diff Against Other Version

Compare the selected version with another version of the element. AccuRev changes the mouse pointer, to prompt you to select the other version. You can click on any version in the Version Browser display. You can also click on a stream or workspace label to indicate the version currently in that stream or workspace.

clicking a stream label indicates the version currently in that stream



### Diff Against File in the Workspace

Compare the selected version with the file in the workspace from which the Version Browser was invoked. This workspace is listed at the bottom on the AccuRev GUI window. This

enables you to invoke a comparison with a file that you have changed in your workspace, but have not yet preserved in the repository with a **Keep** command.

### Send to Workspace

Activate the selected element in the workspace from which the Version Browser was invoked.

### Send to Issue

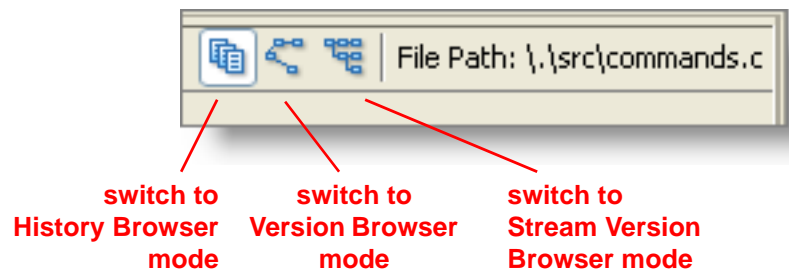
Record the selected version(s) in the change package section (Changes tab) of one or more issue records. The default query of the issues database is executed, and you are prompted to choose one or more of the records selected by the query. You can also create a new issue record, to which the selected version(s) will be sent.

### Send to Issue (specifying basis)

Record the selected version in the change package section (Changes tab) of one or more issue records. You are prompted select a version in the Version Browser display; this version will become the basis version in the change package entry. Then, the default query of the issues database is executed, and you are prompted to choose one or more of the records selected by the query. You can also create a new issue record, to which the selected versions will be sent.

## Quick Switch: History / Version / Stream Version Browsers

In the toolbar of all History Browser, Version Browser, and Stream Version Browser tabs, there are three buttons in the upper right corner. These buttons enable you to perform a “quick switch”, for example to change a Version Browser tab into a History Browser tab.




You can think of the three browsers as the several display modes of a single “Versions” tool. The multiple modes provide different ways to view some or all of the versions of an element.

# The Stream Version Browser

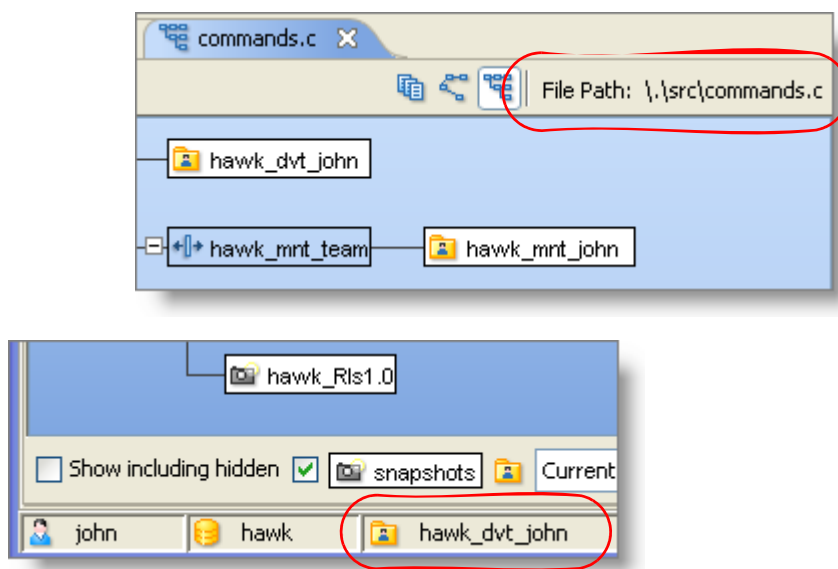
The Stream Version Browser tool is a variant of the StreamBrowser. It displays the stream hierarchy for an entire depot; each stream, snapshot, and workspace in the display represents a version of a specified element.

## Invoking the Stream Version Browser

The Stream Version Browser displays information about one particular element. In a File Browser, select a file or directory element in the details pane. Then click the  **Browse Stream Versions** toolbar button,

## The Stream Version Browser Display

The Stream Version Browser display is almost exactly the same as the StreamBrowser display. But the pathname of the specified element is displayed in the upper right corner of the tab. Also, the workspace that you were in (when you invoked the Stream Version Browser from in the File Browser) is displayed at the bottom of the GUI window.



Each box in the Stream Version Browser display represents the version of the specified element that is currently in that stream, snapshot, or workspace.

Note: a workspace box indicates the version in the workspace stream, which is not necessarily the same as the file in the workspace tree. They differ if the file has **(modified)** status.

## Working in the Stream Version Browser

The Stream Version Browser has many of the same commands as the StreamBrowser for controlling the display. These commands — available in the Stream Version Browser toolbar, in context menus, and in the Actions submenu of the GUI's main menu — include:

### Zoom In

Restricts the Stream Version Browser display to the subhierarchy below the selected stream.

**Zoom Out**

Returns the Stream Version Browser to displaying the depot's entire stream hierarchy.

**Graphical Display**

Shows streams as boxes, in a tree-shaped hierarchy.

**Tabular Display**

Shows streams in a table.

**Graphical/Tabular Display**

Both of the above.

Similarly, the Stream Version Browser includes many of the same commands as the Version Browser, for operating on a selected version of an element. These include:

**Open**

Run the appropriate command on the version, according to its file type.

**View**

Open a text editor on a temporary copy of the currently selected version (text files only).

**Save As**

Copy the currently selected version to another filename.

**Merge From**

Merge the selected version into the version in the workspace from which the Stream Version Browser was invoked.

**Patch From**

Patch the selected version into the version in the workspace from which the Stream Version Browser was invoked.

**Send To Workspace**

Activate the selected element in the workspace from which the Stream Version Browser was invoked.

**Send To Issue**

Record the selected version in the change package section (Changes tab) of one or more issue records. The default query of the issues database is executed, and you are prompted to choose one or more of the records selected by the query. You can also create a new issue record, to which the selected version(s) will be sent.

**Show Difference**

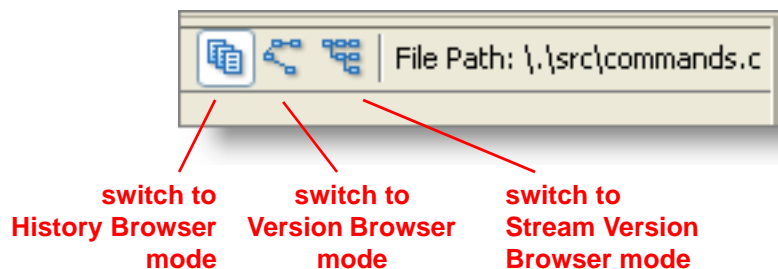
Compare the selected version with another version of the element. AccuRev changes the mouse pointer, to prompt you to select the other version.

## Properties

Displays a message box containing information about the selected version.

## Quick Switch: History / Version / Stream Version Browsers

In the toolbar of all History Browser, Version Browser, and Stream Version Browser tabs, there are three buttons in the upper right corner. These buttons enable you to perform a “quick switch”, for example to change a Version Browser tab into a History Browser tab.



You can think of the three browsers as the several display modes of a single “Versions” tool. The multiple modes provide different ways to view some or all of the versions of an element.





## Using the Change Palette

Usually, the structure of a depot's stream hierarchy determines how versions of an element will be propagated from stream to stream:

- A new version is originally created in some user's workspace stream.
- Later, the version is promoted to the parent stream.
- Still later, the version is promoted from the parent stream to *its* parent.
- And so on, typically all the way up to the depot's root stream.

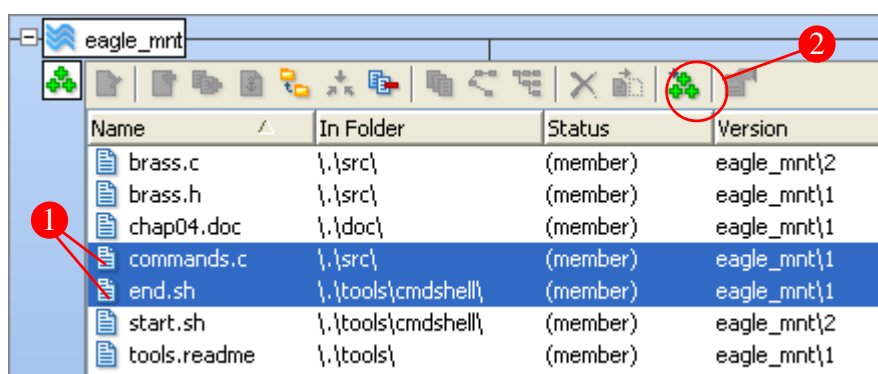
At times, however, you may want to propagate changes directly to some stream other than the parent stream. When you want to go “outside the lines” of the depot's stream hierarchy, the Change Palette is the tool to use.

Using the Change Palette, you can promote active versions from a dynamic stream to another dynamic stream. The Change Palette also detects the need for a merge prior to a desired promotion, and manages the carrying out of such merges. You can also use the Change Palette to send an active version from a dynamic stream to a workspace. This is actually a **Send to Workspace** operation (CLI command: **co**), not a **Promote** operation.

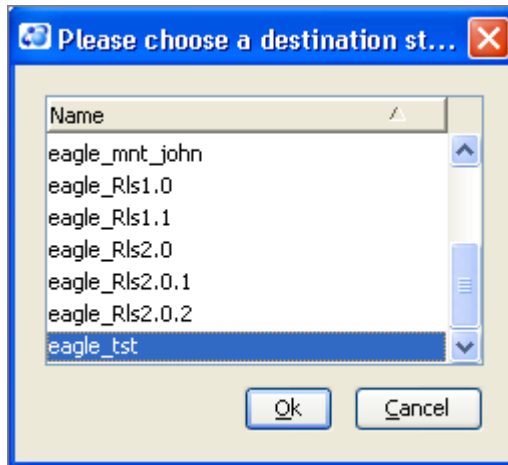
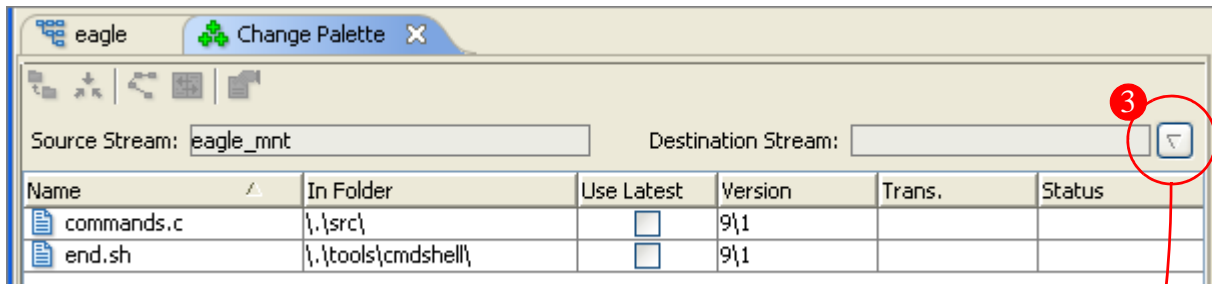
You cannot use the Change Palette to promote versions from your workspace. Instead, you must use the **Promote** command to send such versions to the workspace's backing stream. Also note that a lock can be placed on a dynamic stream, preventing promotion either *from* that stream or *to* that stream.

In the most common case, using the Change Palette is almost as simple as executing a **Promote** command:

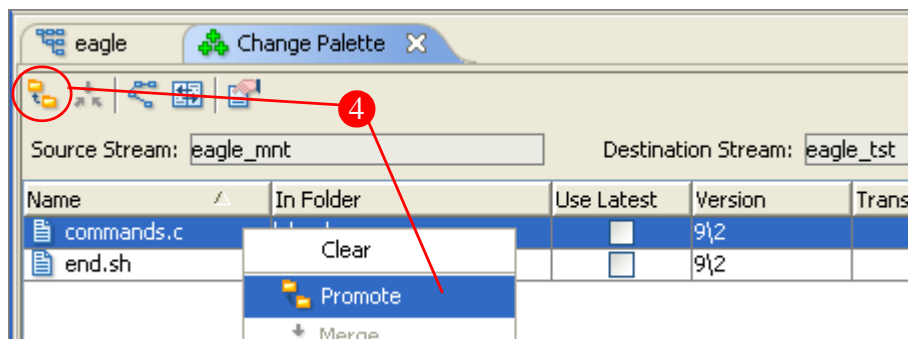
1. Using the StreamBrowser, select the version(s) to be promoted from the Default Group of some dynamic stream.
2. Invoke the **Send to Change Palette** command.



3. In the Change Palette, specify the destination stream or workspace for the version(s).



- Invoke the **Promote** command on the version(s). (If the destination is a workspace, the command is **Send to Workspace** instead of **Promote**.)

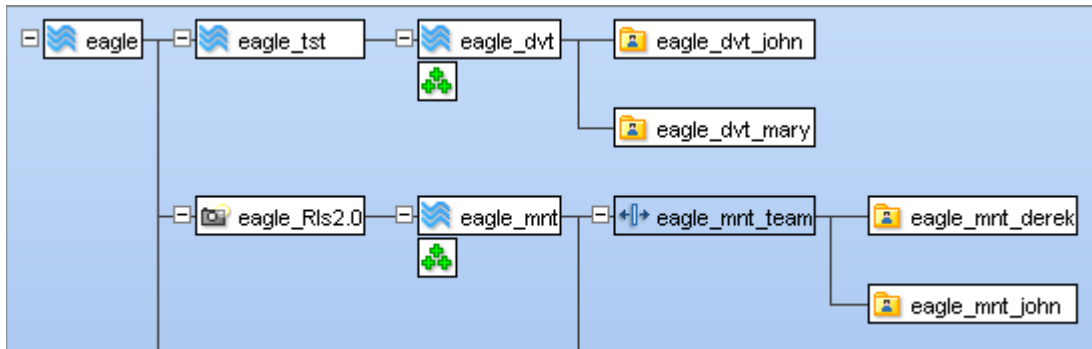


If a version must be merged with the version in the destination stream prior to promotion, the Change Palette displays an **overlap** status flag. You can invoke the **Merge** command from the Change Palette, and also control which workspace the merge will take place in.

An **overlap** can also occur when you choose a workspace as the destination. In this case, instead of merging, you can choose to simply replace the version in the workspace with the version in the source stream.

If a workspace is the destination, you can choose to **Patch** from the selected version, instead of invoking **Send to Workspace** or **Merge**. The **Patch** command takes just the recent changes from the selected version, whereas **Send to Workspace** and **Merge** take all the changes.

The sections below describe all of the Change Palette's capabilities, covering all the use cases. The examples all come from the depot pictured here:





We'll show how to use the Change Palette to propagate changes directly ...

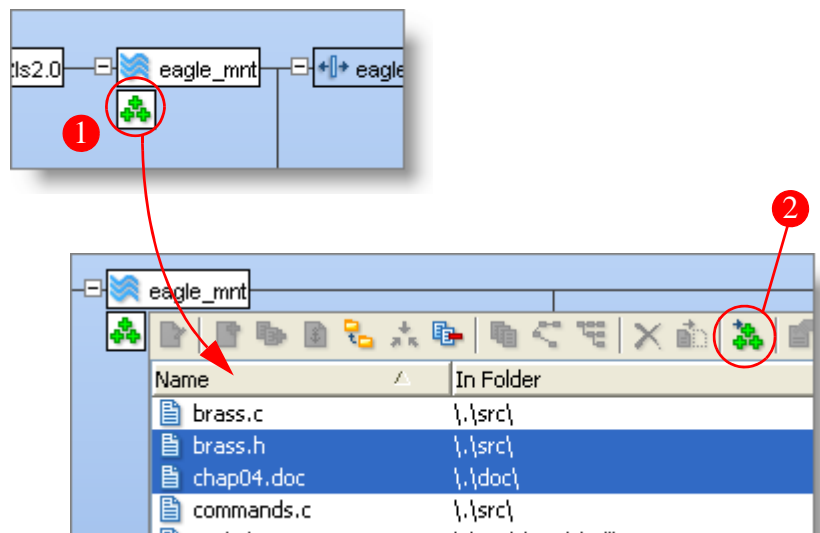
- ... from the **eagle\_maint** stream to the **eagle\_dvt** stream
- ... from the **eagle\_mnt** stream to the **eagle\_dvt\_mary** workspace
- ... from the **eagle\_dvt** stream to the root stream, **eagle** (bypassing the **eagle\_tst** stream)

## Sending Selected Versions to the Change Palette

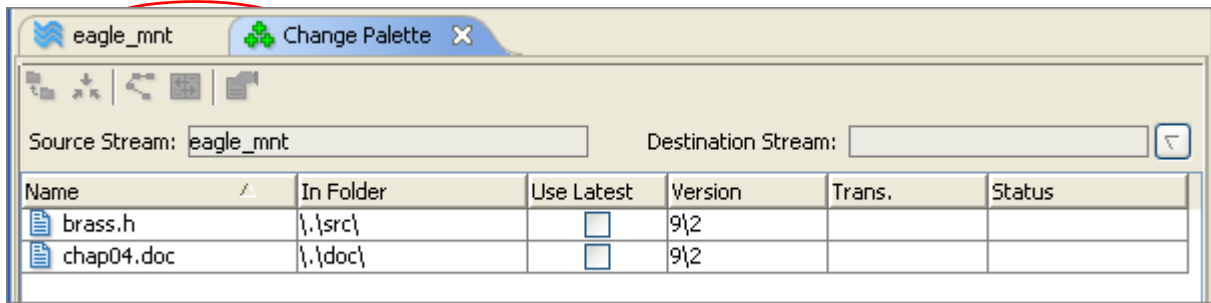
Before using the Change Palette itself, you must specify the version(s) that are to be promoted “outside the lines” of the depot's stream hierarchy. One way is to select the versions in the StreamBrowser, File Browser, or History Browser, then invoke the **Send to Change Palette** command.

Only active versions in dynamic streams can be sent to the Change Palette. That is, a version must be in the default group of a dynamic stream to be eligible for promotion to another dynamic stream (or sending to a workspace) using the Change Palette. The easiest way to select such versions is with the StreamBrowser:

1. To see the contents of any stream's default group, click the  default group control that appears below the stream.
2. In the default-group listing, select the versions you wish to promote, then invoke the  **Send to Change Palette** command, using the toolbar button or the context menu.

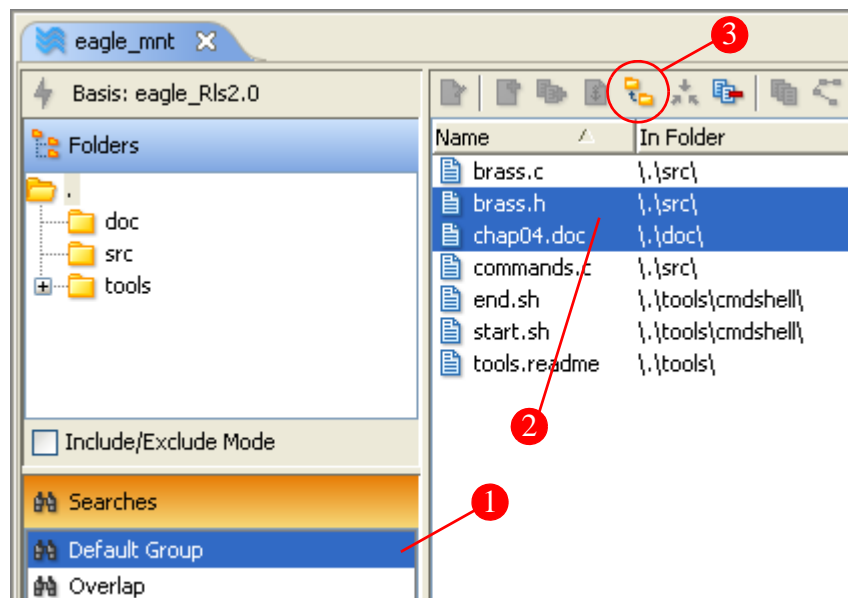


AccuRev opens a Change Palette tab, with an entry for each version you selected.




You can also select the versions to send to the Change Palette using the File Browser. Open a File Browser on a dynamic stream by double-clicking the stream in the StreamBrowser. Then:

1. In the searches pane of the File Browser, select the Default Group search.
2. Select one or more versions from the results of the search.
3. Invoke the **Send to Change Palette** command, using the toolbar button or the context menu

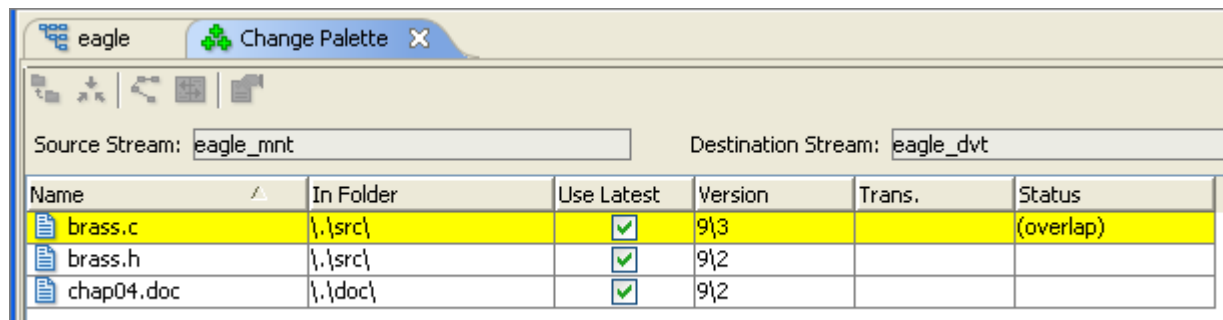


## Letting AccuRev Select the Versions to be Sent

There's another, particularly powerful way to populate the Change Palette. Instead of selecting individual versions from the Default Group of a stream, you can let AccuRev determine the *complete set* of versions in some source stream that have not yet been propagated to some destination stream or workspace. Just drag the source stream's default group control to the destination stream. Alternatively:

1. In the StreamBrowser, right-click the source stream and select **Send to Change Palette** from the context menu. (Or use the  **Send to Change Palette** button in the StreamBrowser's toolbar.)
2. The mouse cursor changes to include the Change Palette icon. Click on another dynamic stream or a workspace to be the destination.

This populates the Change Palette with an entry for each version that needs to be propagated to the destination stream or workspace.



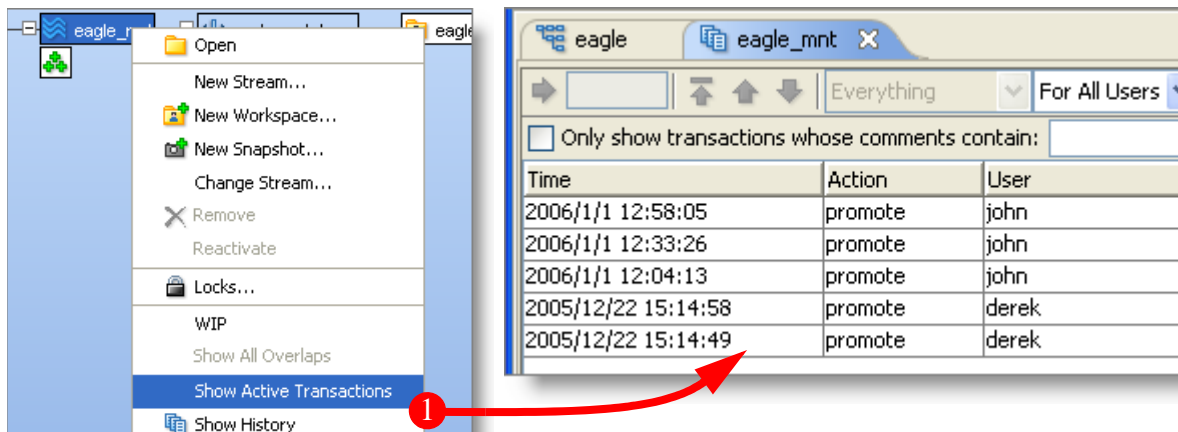
When you populate the Change Palette in this way, AccuRev automatically selects **Use Latest** for each version. A check in this column indicates that when you proceed to promote the element to the destination stream, AccuRev should use the version that is in the source stream at the time you invoke **Promote**. Given the indefinite time lag, this might be different from the version, listed in the Version column, that **Send to Change Palette** initially selected. Deselecting **Use Latest** for an element guarantees that the version listed in the Version column will be the one propagated to the destination stream.

Note: in some cases, **Use Latest** propagates a version that is “in the source stream”, but not “active in the source stream”, to the destination stream. This occurs when the selected version gets promoted out of the source stream between the time you invoke **Send to Change Palette** and the time you invoke **Promote**. The version propagated to the destination stream is one that the source stream is inheriting from a higher-level stream at the time of the **Promote**.

## Populating the Change Palette from the History Browser

Yet another way to send a stream’s active versions to the Change Palette involves the History Browser. The command **Show Active Transactions** finds all the **Promote** transactions that “activated” a stream’s currently-active versions; it opens a History Browser tab to display these transactions. For example:

1. Invoke the **Show Active Transactions** command on stream **eagle\_mnt**. A History Browser tab opens, showing a set of transactions — in this example, five of them. These are **Promote** transactions that created versions in **eagle\_mnt**; the set includes only those transactions with at least one version still active in the stream.



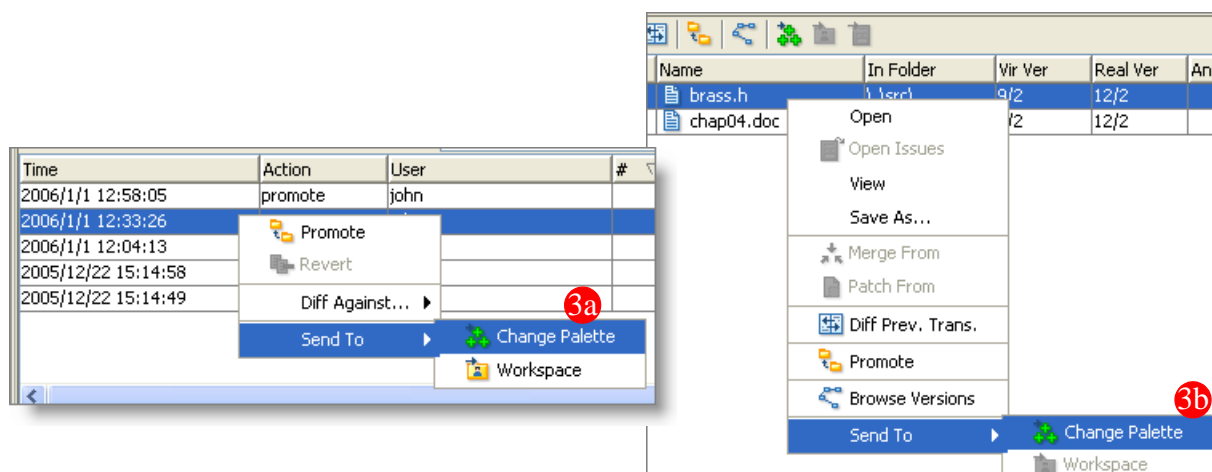
2. Select one of the transactions to see which versions it “activated” in the stream — in this example, transaction #138 created version 9/2 of file **brass.h** and version 9/2 of file **chap04.doc**.

2006/1/1 12:58:05	promote	john	142
2006/1/1 12:33:26	promote	john	138
2006/1/1 12:04:13	promote	john	131
2005/12/22 15:14:58	promote	derek	44
2005/12/22 15:14:49	promote	derek	42

Status	Name	In Folder	Vir Ver	Real Ver	Ancesto
(member)	brass.h	.\src\	9/2	12/2	
(member)	chap04.doc	.\doc\	9/2	12/2	

3. Use the context menu of one of the transactions to send all its active versions to the Change Palette. Alternatively, select one or more of the transaction’s versions in the lower pane, and use the context menu to send all the selected versions to the Change Palette.



Note: one or more of the versions in an “active transaction” may no longer be active in the stream. This indicates that the version has already been promoted to the parent stream. You cannot send such

versions to the Change Palette. When all the versions in an active transaction have been promoted, the transaction will no longer appear in the Show Active Transactions listing. Note also that a version can become inactive through a purge (**Revert to Backed Version**) instead of a promotion.

Status ▾ 1	Element ▴ 2	Vir Ver
(member)	\\.\doc\chap01.doc	4/3
	\\.\doc\chap02.doc	4/2

no (member) status flag: this version of chap02.doc is no longer active in this stream

## Working in the Change Palette

After you’ve sent a set of versions to the Change Palette, you can proceed to propagate them to the destination stream or workspace. Here’s a summary of the procedure:

1. If necessary, specify the destination stream to which the version(s) are to be propagated.
2. If necessary, merge the source and destination versions of one or more of the elements.
3. Perform the **Promote** or **Send to Workspace** command. If the destination is a workspace, the **Patch** command is an alternative.

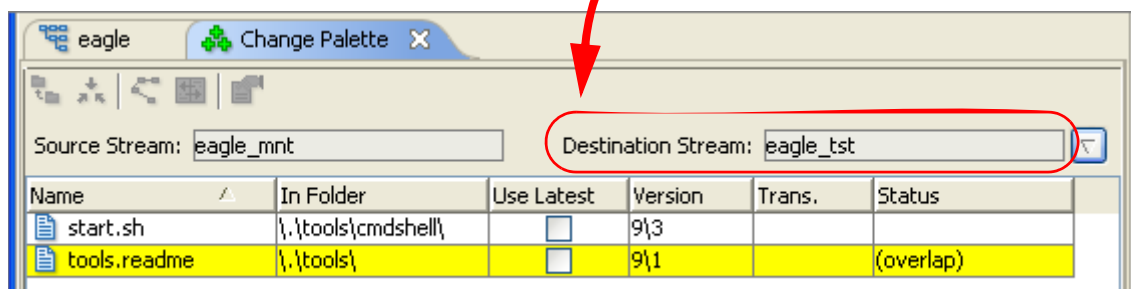
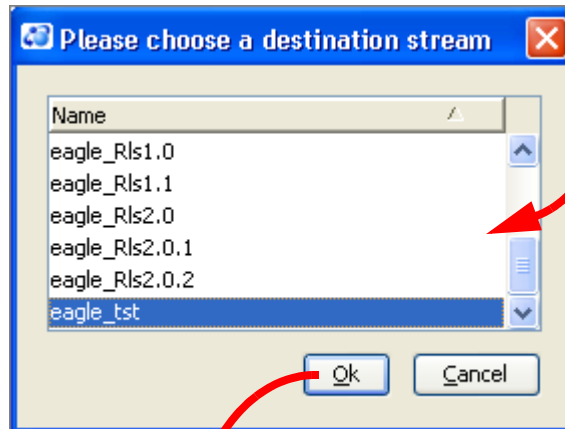
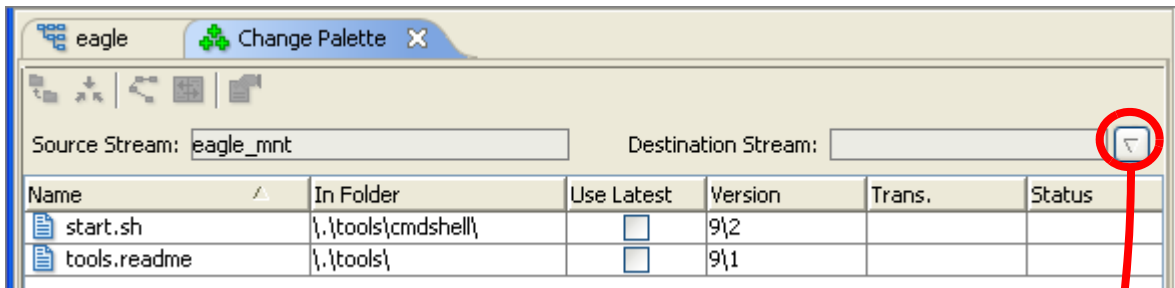
The following sections explore the details of this procedure.

## Specifying the Destination Stream

Note: you can skip this step if you used the procedure described in section *Letting AccuRev Select the Versions to be Sent* above, because you’ve already specified both the source and destination.

At this point, the **Source Stream** field shows the stream in which you invoked the **Send to Change Palette** command. Now, you must indicate a destination stream or workspace. Click the arrow control to the right of the **Destination Stream** field. A dialog box appears, containing a list of dynamic streams and workspaces; choose one of them to be the destination.





## Merging the Source and Destination Versions (If Necessary)

When you specify a dynamic stream as the destination, AccuRev determines for each element whether a merge is required between the source-stream version and the destination-stream version. If so, it highlights the entry in yellow and places an **overlap** indicator in the **Status** column.

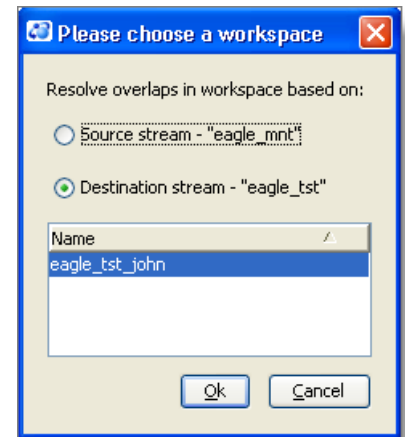
Before you can promote the source-stream version to the destination stream, you must resolve the overlap status by performing a merge. The **Merge** command combines the contents of the source-stream version with the contents of the destination-stream version, and creates a new version of the element with the combined (“merged”) contents.

## Selecting a Workspace for Performing Merges

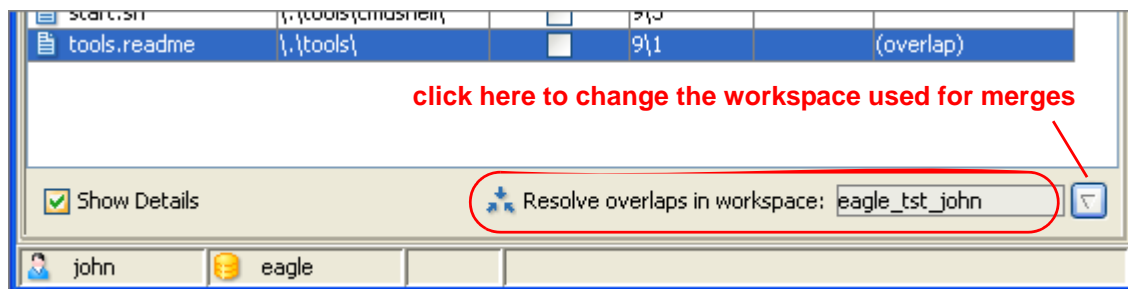
Where does the merged version get created? It can’t be either be in either the source or destination stream, because these are *dynamic* streams — all new versions of AccuRev elements must be

originally created in a *workspace* stream. For the **Merge** command to proceed, a workspace belonging to you must be attached to either the source stream or the destination stream.

The first time you invoke **Merge** in the Change Palette, AccuRev prompts you to establish a merge workspace, suggesting one that is attached to the destination stream. You can also choose a workspace attached to the source stream.




Thereafter, the merge workspace setting is displayed at the bottom of the Change Palette. You can change this setting at any time; this enables you to use different workspaces to merge different Change Palette entries.



What if *no* workspace exists that belongs to you and is attached to either the source or destination stream? In this case, you must click **Cancel** in the dialog box, and do either of the following:

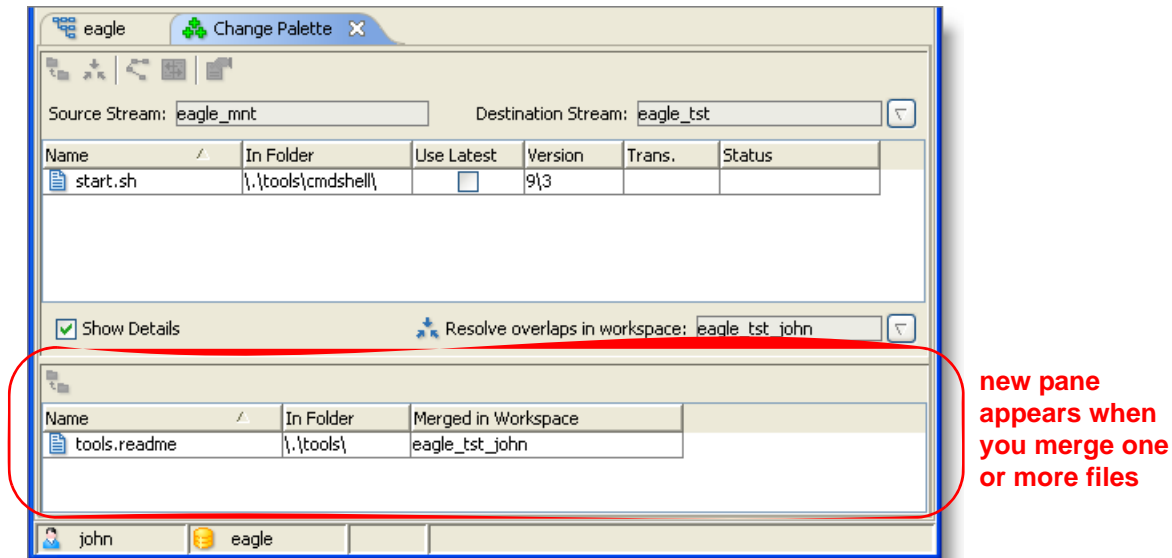
- Create a workspace attached to one of the streams. (In the StreamBrowser, right-click the stream and select **Create New Workspace** from the context menu.)
- Re-parent one of your existing workspaces: in the StreamBrowser, drag-and-drop the workspace from its current backing stream to the source or destination stream. Note that you should **Update** a reparented workspace before using it to perform Change Palette merge work.

## Performing the Merges

To proceed with merging, select one or more of the overlap-status files in the Change Palette and click the  **Merge** toolbar button. Alternatively, right-click the line-item(s) and select the **Merge** command from the context menu. You can merge the overlap-status files all at once or one at a time. Perform the merge(s) in the standard way. (See [Viewing and Resolving Content-Level Conflicts](#) on page 125.)


When you've finished merging the file, several things occur:

- The Merge tab disappears, so that the Change Palette tab appears again.
- A new version of each merged file is created in the selected workspace; the new version contains the merged contents.
- A new pane opens at the bottom of the Change Palette tab, displaying these new workspace version(s).



At any time, you can promote one or more of these merged versions, as described in the next section.

## Performing the Promotions

You can **Promote** one or more files from the Change Palette's upper pane or lower pane (but not both panes at once), using the  **Promote** tool button or the files' context menu. Files in the upper pane can be promoted if they do not have overlap status. All files in the lower pane can be promoted.

Note: Files promoted from the upper pane go to the designated destination stream. Files promoted from the lower pane go to the backing stream of the workspace in which the merge was performed. Depending on the situation, this may be the source stream or the destination stream.

This means that when you merge using a workspace based on the source stream, you need to promote the merged version again — from the source stream to the destination stream — to achieve your original goal of propagating changes to the destination stream.

## CLI Commands Related to the Change Palette

The following AccuRev CLI commands implement the various Change Palette capabilities described in this chapter:

- The **mergelist** command implements a stream's **Send to Change Palette** command in the StreamBrowser. (This is the command that determines *all* the changes that need to be propagated from one specified stream to another.)
- The **merge** command implements the **Merge** GUI command — both in the Change Palette and in the File Browser. The version in the “Resolve overlaps in ...” workspace is one of the merge contributors; the **-v** option specifies the other merge contributor.
- The **promote** command implements the **Promote** GUI command — both in the Change Palette and in the File Browser. The **-s** and **-S** options specify the source and destination dynamic streams.

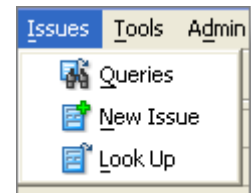


# Working with Issue Records

This chapter discusses usage of AccuWork, the AccuRev issue-management facility from the viewpoint of the day-to-day user. The topics include:

- *Accessing a Particular Issues Database*
- *Creating a New Issue Record*
- *Viewing and Modifying an Existing Issue Record*
- *Printing an Individual Issue Record*
- *Using AccuWork Queries*
- *Printing Query Results*

This chapter covers the commands on the **Issues** menu: **New Issue**, **Look Up**, and **Queries**. See *Designing Issues Databases and Edit Forms: The Schema Editor* on page 189 for a discussion of the AccuWork Schema Editor, the tool for creating issues databases and their edit forms.

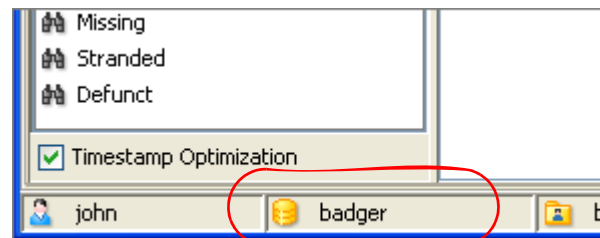


## Accessing a Particular Issues Database

In the AccuRev GUI window, you can access any number of issues databases, using different tabs. The name of the depot you're using in the current tab is displayed at the bottom of the window.

When you invoke the AccuWork **New Issue** or **Look Up** command, it applies to the current depot — the one being used by the current tab.

(If there is no current depot, AccuRev prompts you to select one.) When you invoke the **Queries** command, AccuRev prompts you to select a depot.



## Creating a New Issue Record



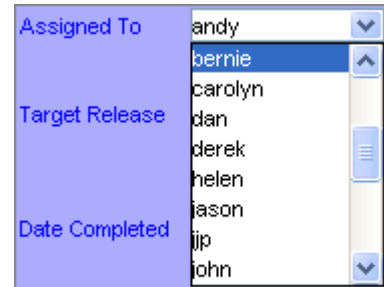
Use the **New Issue** menu command or toolbar button to open an empty edit form. The form is a new tab in the GUI's tabbed display, labeled "New Issue". Depending on the design of the issues database, you'll notice some or all of the following features:


- Certain fields have already been initialized with particular values.
- Certain field-labels are in a contrasting color (by default, red), indicating that they are required fields. You cannot save a new issue record until a value appears in all required fields. Some of those fields may have gotten their values by being initialized.
- You can't enter a value in the **Issue** field. This is the unique issue-number for this particular issue record. AccuWork fills in this field when you save the issue record.

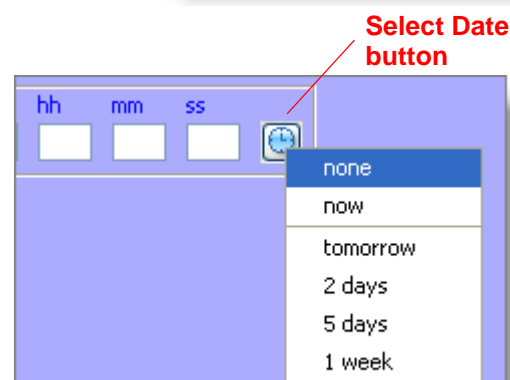
Enter values in the fields, using the mouse or **Tab** and **Shift-Tab** to move from field to field. Within a multiple-line text field, the **Tab** key inserts a TAB character, rather than jumping to the next field.


With several kinds of fields, you can enter a value in some way other than simple typing:

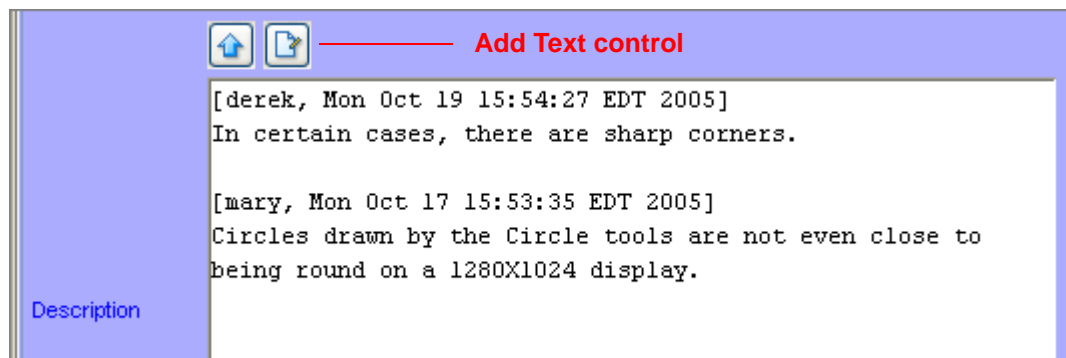
- **list-box** — a multiple-choice box works in the same way as on Web forms. You can click on the arrow control to see some or all the choices (maybe with a scroll bar). You can use arrows to scroll through the choices, without having to open the list. Typing a letter advances to the next item that begins with that letter.



- **timestamp** — You can fill in the various parts of a timestamp field manually, or use the  **Select Date** button to display choices that fill in the fields automatically. Once these fields are filled in, you can go back and revise them.

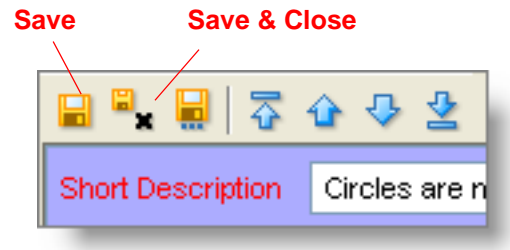


- **log** — A log field is a variant of the multiple-line text field. You can type directly into such a field, or you can click the  **Add Text** control, which pops up a separate window. The text that you type (or paste) into this window is added to the current contents of the field, along with a “log stamp” that incorporates your AccuRev username and the current time.

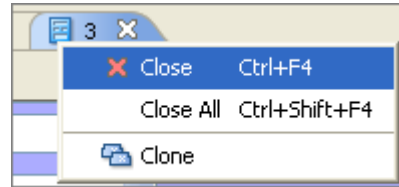


The edit form will have multiple pages, which appear as subtabs within the overall “New Issue” tab. And typically, there’s a “header section”, which always remains visible as you switch from page to page. Use the mouse to switch among multiple pages.

When you're ready (have you filled in all the required fields?), click the **Save** button or the **Save & Close** button in the toolbar above the New Issue form. AccuWork assigns an issue-number to the new record and stores it in the depot. With **Save**, the new record remains on-screen; the issue-number replaces the "New Issue" label and appears in the **Issue** field. With **Save & Close**, the tab containing the edit form disappears.



At any time, you can close an edit-form tab as you would any GUI tab: right-click the tab's title, then select **Close** from the context menu. If AccuRev Look And Feel is enabled (**Tools > Preferences**), you can close the tab using the "X" icon on the tab control itself.

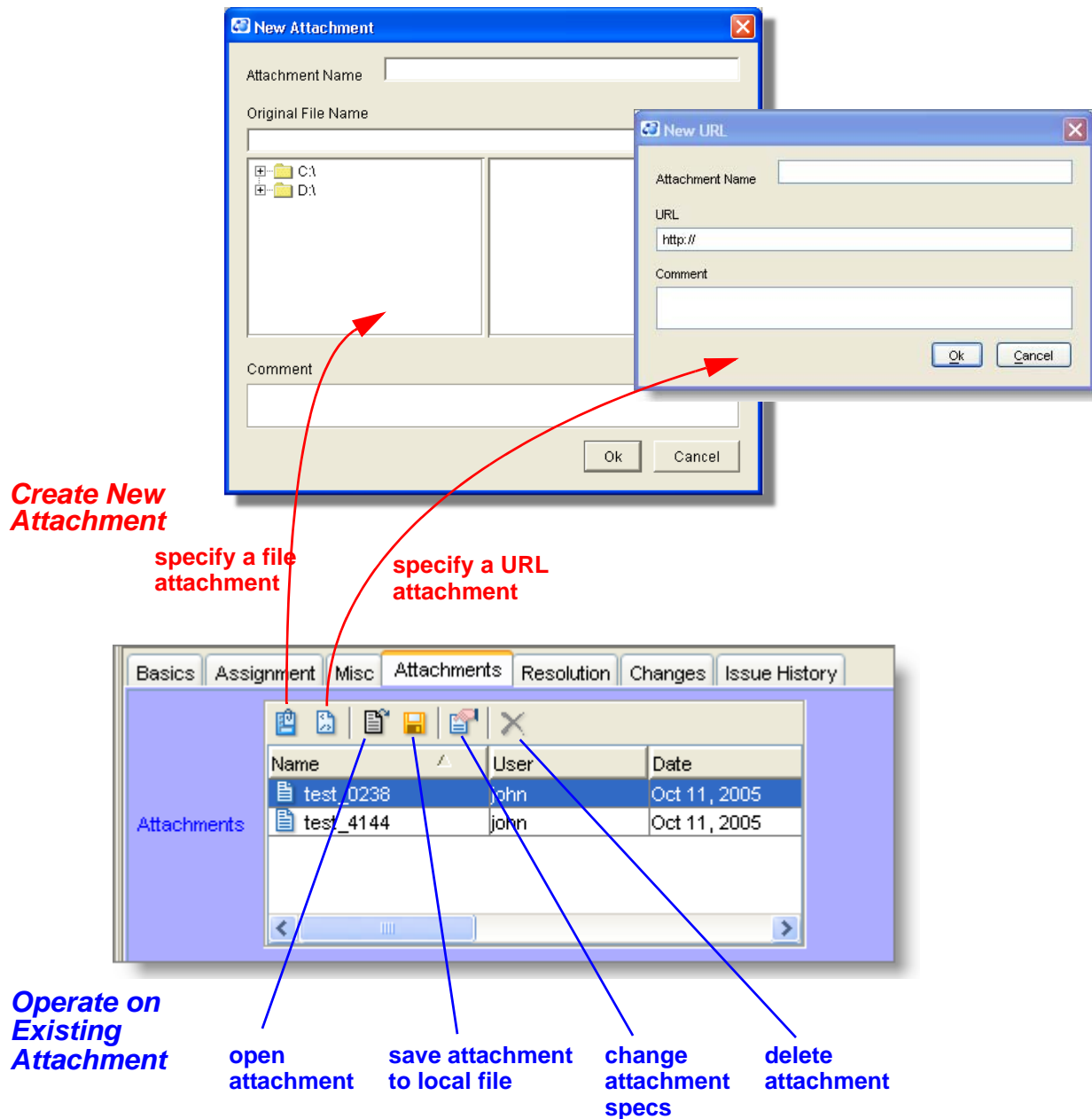


If you want to create another issue record, use the **New Issue** menu command or toolbar button again.

## Creating Attachments to an Issue Record

An edit form can contain one or more attachment fields. In each such field, you can specify one or more files and/or Internet addresses (URLs) to be attached to the current issue record. AccuWork displays the data as an attachments table.





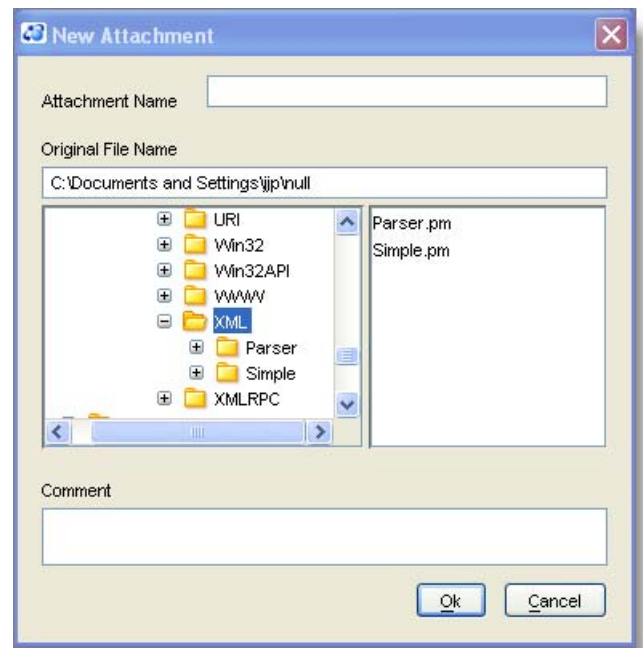
In addition to specifying the location of a file or Internet resource, you enter a Name and optional Comment. AccuWork automatically fills in your username, the date, and the size of the attached file. (Internet URLs get assigned a size of 0.) If the edit-form field is not large enough to show all these attachment parameters, use the scroll bar to see all the data. You can also resize and rearrange the columns of an attachments table.

An attachment field includes its own toolbar, with these buttons:

## New Attachment

Define a new file attachment for this issue record. A file selection dialog box appears, in which you can specify one or more files, a name for the attachment, and a comment string.

You can specify multiple files at once — each one becomes a separate file attachment. In this case, the Attachment Name input field is disabled; each filename is automatically assigned as the attachment name. The comment string that you specify is assigned to each file attachment.



## New URL

Define a new attachment to be the address (URL) of an Internet resource. A dialog box appears, in which you specify a URL, a name for the attachment, and a comment string. AccuWork helps out by placing the string **http://** in the URL field. You can erase this if you want to specify a location accessed by another Internet protocol, such as **ftp://**.

## Open Attachment

Open the existing attachment that is currently selected, using the appropriate program.

## Save Attachment As

Create a copy of the currently selected attachment on the client machine.

## Properties

Launch a Properties window, displaying the definition of the currently selected attachment. You can use this window to change the attachment's Name or Comment value.

## Delete Attachment

Remove the attachment from the issue record (and from the depot).

When you save the issue record, each file is copied to the depot, so that the data always remains available through the issue record, even if an original file is deleted.

## Canceling Creation of a New Issue Record

If you decide not to create a new issue record, just close the edit form (as described above) without ever clicking the **Save** button.

## Issue Records and Transactions — the Issue History Page

When you create a new issue record (or update an existing record), a **dispatch** transaction is written to the depot's transaction log — the same log that records AccuRev transactions, such as **keep** and **promote**. It's important to distinguish the depot's transaction log (which is a database in its own right) from the AccuWork issues database stored within the depot.

AccuWork automatically adds an **Issue History** page to each edit form. This page shows how an issue record has changed over time. It displays all the **dispatch** transactions for an issue record, including the initial **Save**. Each row of the table shows the change to one field made by a subsequent **Save**.

Tr. ▾	User	Date	Field	New Value	Old Value
83	john	Jan 9, 2006 1:40:27 PM	targetRelease	rel2.0	
83	john	Jan 9, 2006 1:40:27 PM	type	defect	
83	john	Jan 9, 2006 1:40:27 PM	assignedTo	dan	
83	john	Jan 9, 2006 1:40:27 PM	submittedBy	helen	
83	john	Jan 9, 2006 1:40:27 PM	state	WIP	
83	john	Jan 9, 2006 1:40:27 PM	dateAssigned	May 25, 2004 8:55:25 PM	

If you double-click any row of the Issue History table, AccuWork opens a read-only edit form, showing the state of the issue record at the time of that particular **Save**.

## The server\_dispatch\_post Trigger

The act of saving a new issue record (or updating an existing record) may cause a user-defined procedure to be performed on the AccuRev Server machine. This is implemented through AccuRev's trigger facility. Typically, a trigger script sends an email message to one or more interested parties — for example, the user to whom a bug has been assigned.

For more information, see *Post-Operation Triggers* on page 276 of the *AccuRev Administrator's Manual*.

## Viewing and Modifying an Existing Issue Record

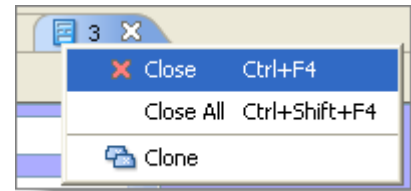
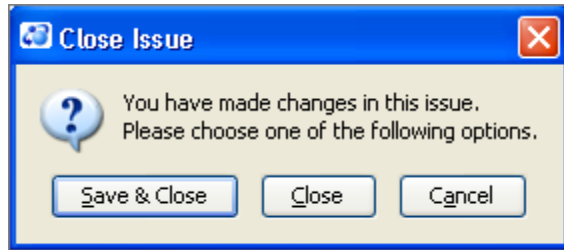


The easiest way to retrieve an existing issue record is through its unique issue-number. Use the **Look Up** menu command or the **Open Issue** toolbar button. AccuWork prompts you to enter an issue-number, retrieves the record, and displays it in the edit form. You can also view and edit issue records that have been selected by a query; see *Query Results Pane* on page 178.

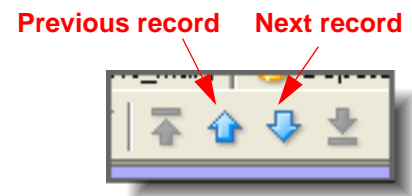
Note: issue-numbers are unique within a given depot. Several different depots might have an issue-record numbered 382. Make sure you've opened the right depot!

If you wish, modify one or more fields, then click the **Save** or **Save & Close** button. Required-fields restrictions apply when you're modifying an existing issue record, just as they do when you're creating a new one.

If you don't want to modify the issue record, or if you change your mind after modifying some fields, just close the GUI tab containing the edit form without clicking the **Save** button. AccuWork asks for confirmation, to make sure that you don't mistakenly discard work that should be saved:




While you're viewing or modifying an issue record that you've displayed with **Look Up**, browse arrows are enabled in the edit form's toolbar. This makes it easy to view a set of consecutive issue records. If you've made some changes to an issue, AccuWork prompts you to save or discard those changes before switching to the previous or next one.

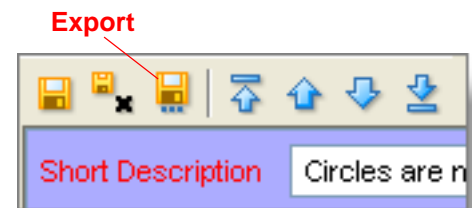


As described in section *Issue Records and Transactions — the Issue History Page* on page 172, the updating of an issue record is recorded as a **dispatch** transaction in the depot's transaction log.

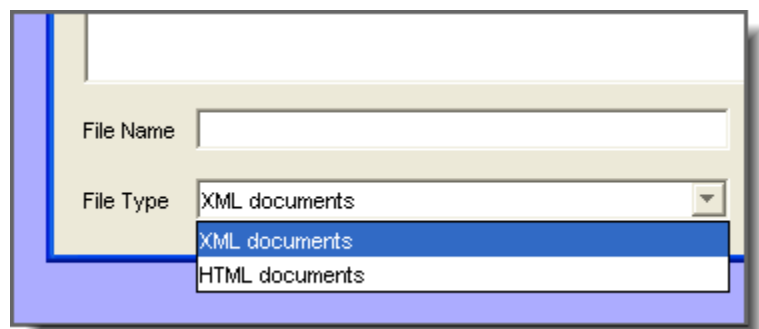
## Printing an Individual Issue Record

At any time when you're using an edit form, whether creating a new issue record or modifying an existing one, you can "print" it. For AccuWork, printing means "publish to the Web" — which means either "create an HTML file" or "create an XML file".

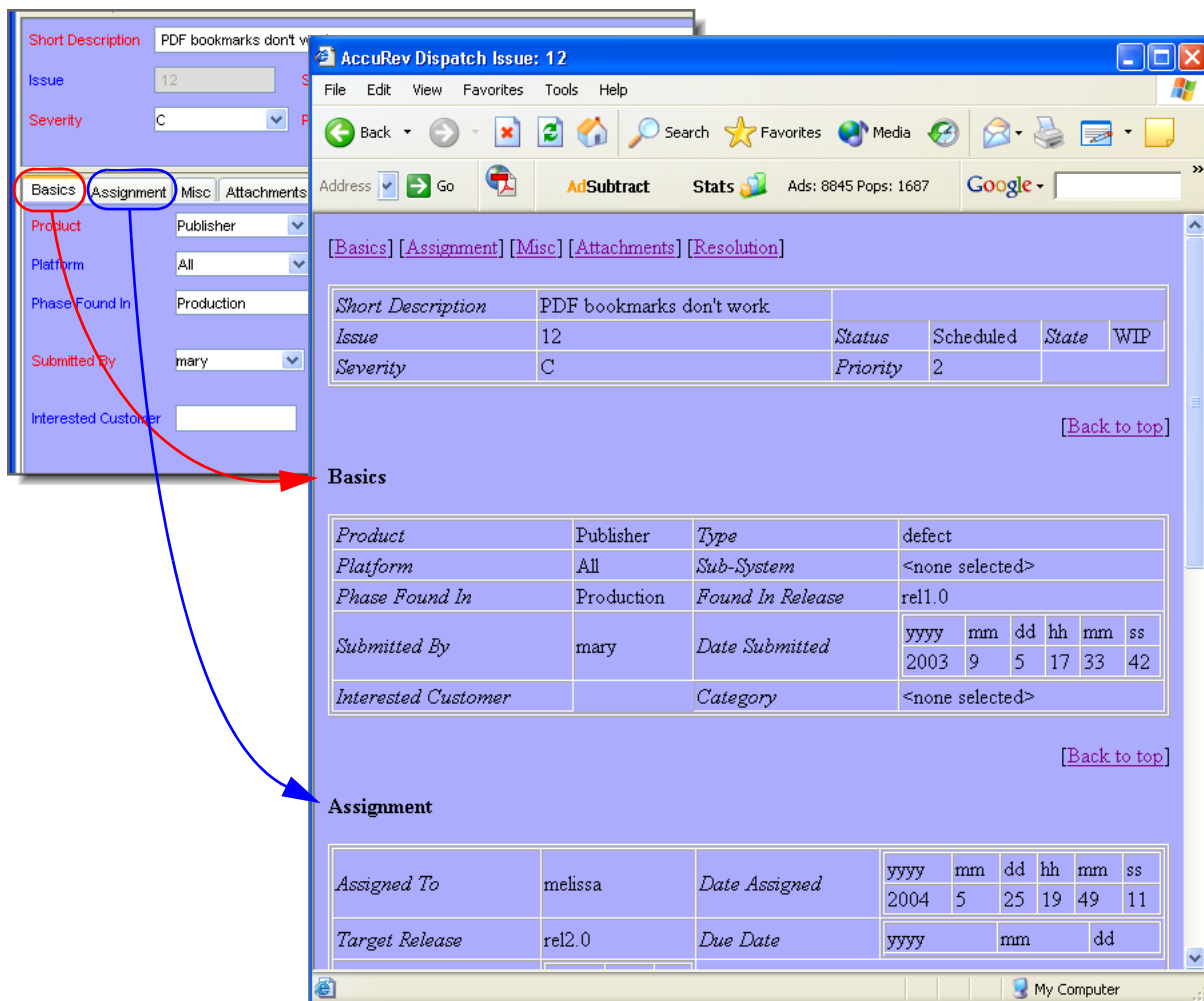
- Click the  **Export** button in the form's toolbar.



- Select the file type in the File Chooser window that appears.
- Specify a pathname for the export file, using the navigator and the File Name input field. (You don't need to specify the **.html** or **.xml** suffix — AccuWork adds it automatically.)



When creating an HTML file, AccuWork outputs all the content of the issue record and approximates the form layout, too. Even if the edit form has multiple pages, you need to “print” only once. All of the pages are combined into a single HTML or XML file:



On Windows machines, AccuWork automatically invokes a Web browser on the HTML file it creates. If you wish, resize the browser window to optimize the look of the issue record. HTML documents automatically adjust to changes in window width. Then, use the browser’s print command to create hardcopy of the issue record.

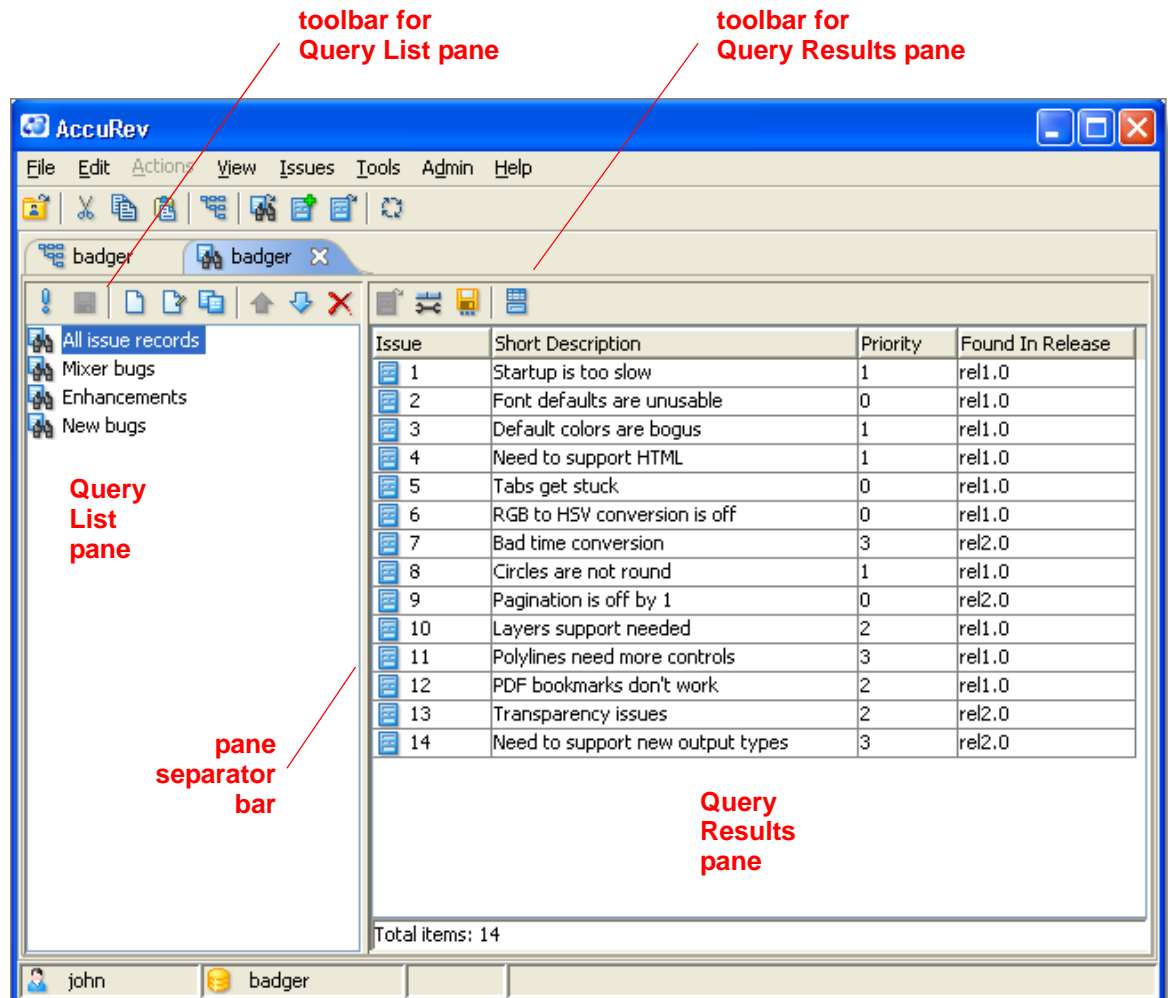
## Using AccuWork Queries

Retrieving one issue record at a time is useful in many situations. But robust issue-management systems must also support queries, which retrieve groups of records according to user-defined selection criteria. AccuWork has a point-and-click interface for creating and editing queries; it enables you to create simple queries quickly, and to create sophisticated queries in a straightforward, reliable way.


By default, the queries that you create are private queries, which cannot be seen by other users. You can declare any query to be a public query. Such queries are visible to all users, who can use and copy them, but cannot modify your original.



Use the **Issues > Queries** command or the **Queries** toolbar button to open a new Queries tab in the GUI window. The Queries tab includes two panes, each with its own toolbar:







- The **Query List** pane lists the names of all your existing queries. (Each AccuRev user has his own set of queries; there are no “global” queries available to all users.) Using the toolbar in this pane, you compose new queries, view/revise/rearrange/execute existing queries, and delete queries.
- When you execute a query, the set of issue records selected by the query are displayed as a results table in the **Query Results** pane. If you’ve set a default query, it’s executed automatically when you open the Queries tab. AccuWork remembers query results as long as the Queries tab stays open. It’s easy to browse through the results of several queries, switching back and forth instantly between different queries’ results tables.

- If you click the  **Show Issue Form** button in the Query Results pane's toolbar, a third pane appears. This pane is a fully functional edit form, which you can use to view and modify the issue record currently selected in the Query Results pane. A user preference (**Tools > Preferences**) causes this pane to appear automatically when you open a Queries tab.

You can use the separator bar between the panes to adjust their relative size. And remember that the GUI window itself is resizable, too.

## Typical Workflow



Here's a typical workflow for creating and using a query:

1. Create a new query: click the  **New Query** button to open the Edit Query window with a blank query, assign the query a name, and specify one or more clauses that select issue records.
2. In the Query Results pane, select click the  **Setup Columns** button and select certain fields to appear in the results table.
3. Click the  **Run Query** button to execute the query, loading data from selected issue records into the results table.
4. Optional: save the results table in HTML format; use a Web browser to view and print the table.
5. Use the  **Save All** button on the Queries toolbar to save the query in the depot, for future use. Each user's queries are stored separately.

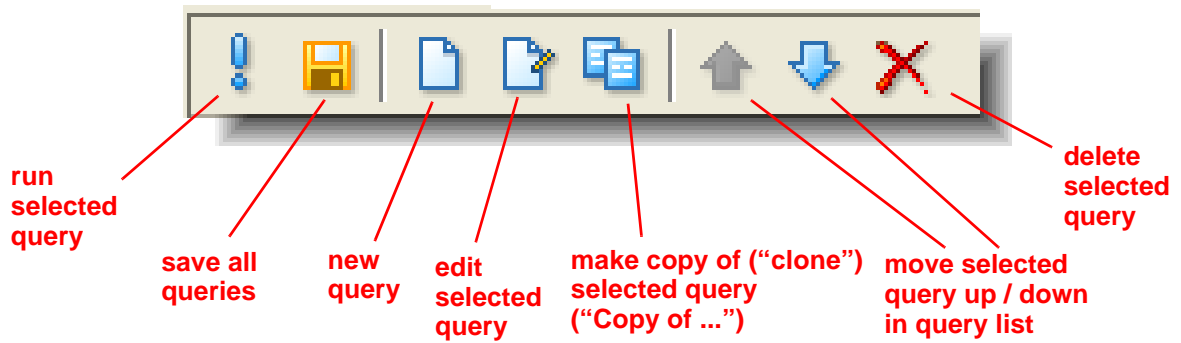
You can refine a query as much as you want, changing the record-selection criteria and the fields that appear in the results table.

The following sections describe the components of the Queries tab in more detail.

## Query List Pane

The Query List pane contains a list of the private queries you've defined for the current depot (issues database), along with any public queries defined by you and/or other users. The listing is not alphabetical — whenever you create a new query, it's simply added to the end of the list. You can rearrange the order of the queries using the toolbar buttons  and . Your queries (both private and public) are always listed above other users' public queries.

You can invoke the following commands on the currently selected query in the Query List pane — either through its context menu or using the Query List toolbar:



## Run query

Executes (or re-executes) the selected query, displaying the results in the Query Results pane.

Whenever you select a particular query in the Query List pane, AccuWork displays in the Query Results pane the most recent results of running that query. (The cache of previous query results is cleared when you close the Queries tab.) You can update a results table to reflect recent AccuWork transactions by clicking the **Run Query** button on the Query List toolbar.

You don't need to click the **Run Query** button when you revise a query in the Edit Query window — AccuWork automatically executes the revised query and updates the results table.

## Save all queries

Save all your private queries. (There is no way to save a single query.) Your queries are stored in the AccuRev repository within the depot directory, in an XML-format file:

```
.../storage/<depot-name>/dispatch/config/user/<principal-name>/query.xml
```

The *<principal-name>* directory in this pathname causes the private queries for each user to be stored separately.

Although your queries are stored in the depot, they are not version-controlled in the way AccuRev files are. For example, there is no command that displays or reinstates your queries as you saved them two days ago.

## New query

### Edit query

Create a new query or revise an existing one. See *The Edit Query Window: Creating and Revising Queries* on page 179.

## Clone query

Create a private query that is a copy of the selected private or public query. The new query is initially named “Copy of ...”, but you can change the name.



### Move query up

### Move query down

Change the position of the selected query in the Query List pane. Note that your own queries (both private and public) are always listed above other users' public queries.

### Delete query

Use the **Delete Query** button to remove the currently selected query. The deletion does not take effect until you save your queries. If you close the Queries tab without saving your queries, the “deleted” query will still exist the next time you open the Queries tab. This may or may not be the right thing to do: closing the Queries tab without saving also discards changes you've made to other queries since the most recent save.

### Set as Default

### Disable as Default

Designates the query to be — or to stop being — the default query for the current issues database. The query name is redisplayed in ***bold-italic*** to indicate that it's the default query.

The default query is executed whenever you open a new Queries tab. It is also executed by the transaction-level integration between AccuRev's configuration management and issue management capabilities. See *Integrations Between Configuration Management and Issue Management* on page 220. And it is executed by the **Send to Issue** command

A query loses its status as the default query when you select **Disable as Default** from its context menu, or when you select another query as the default.

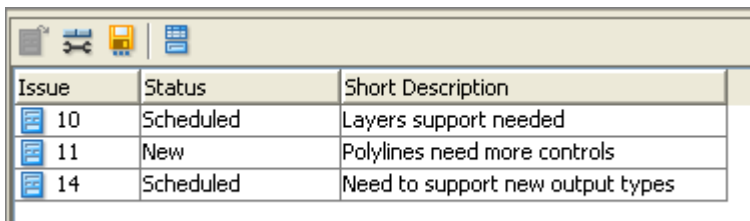
### Set as Private

### Set as Public

Changes a query that you created from public to private, or vice-versa.

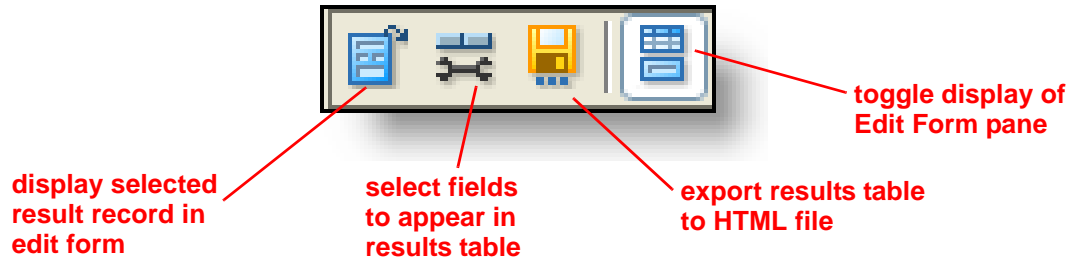
## Query Results Pane

The Query Results pane displays the results of a query as a results table. Each row of the table displays one issue record; each column displays a particular issue-record field.

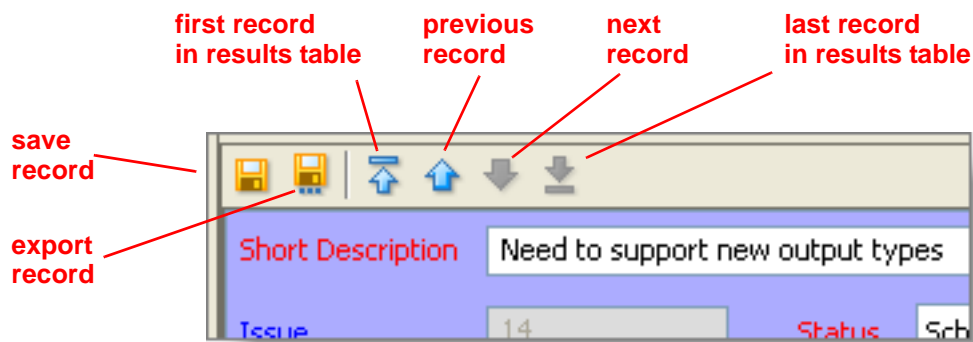


Issue	Status	Short Description
10	Scheduled	Layers support needed
11	New	Polylines need more controls
14	Scheduled	Need to support new output types

Using the mouse, you can rearrange and resize the table's columns. You can also specify a single-level or multiple-level sort order for the table's rows. Additional operations are available through the Query Results pane's toolbar:





- **Open Issue:** Open an edit form to view/revise the selected issue record in the results table.
- **Setup Columns:** Change which columns (fields) appear and their order
- **Export Table:** Create an HTML file containing the entire contents of the results table.
- **Show Issue Form:** Toggles whether an Edit Form pane appears below the Query Results pane. The currently selected record in the Query Results pane is loaded into the Edit Form pane. Toolbar buttons above the edit form enable you to browse some or all of the other records in the results table. You can modify and save an issue record, as described earlier in this chapter; likewise, you can export an issue record to an HTML or XML file.

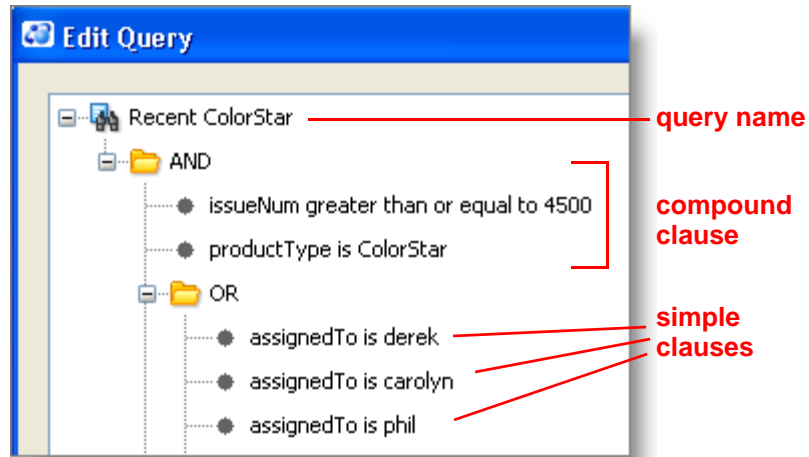


The **Open Issue** and **Setup Columns** commands are also available on the context menu of individual records in the Query Results pane. The context menu also contains the **Remove Column** command, which provides a quick way to revise the structure of the results table: AccuWork removes column that you right-click on.

## The Edit Query Window: Creating and Revising Queries

The Query List pane's toolbar includes  **New Query** and  **Edit Query** command buttons. These commands open an Edit Query window, in which you compose a new query or revise an existing one. A query is displayed as a hierarchy, which you navigate using the familiar expand/collapse controls. The hierarchical organization is a natural fit, because each query is, itself, a hierarchy of simple clauses and compound clauses:

- The first level contains the query's name.
- A simple clause, such as “value of the **assignedTo** field is **mary**”, can live at any level.
- A compound AND clause (*cls-a* AND *cls-b* AND *cls-c* AND ...) consists of the operator AND at one level of the hierarchy and any number of clauses *cls-a*, *cls-b*, *cls-c*, etc. at the next sublevel.
- Compound OR clauses are structured hierarchically in exactly the same way.




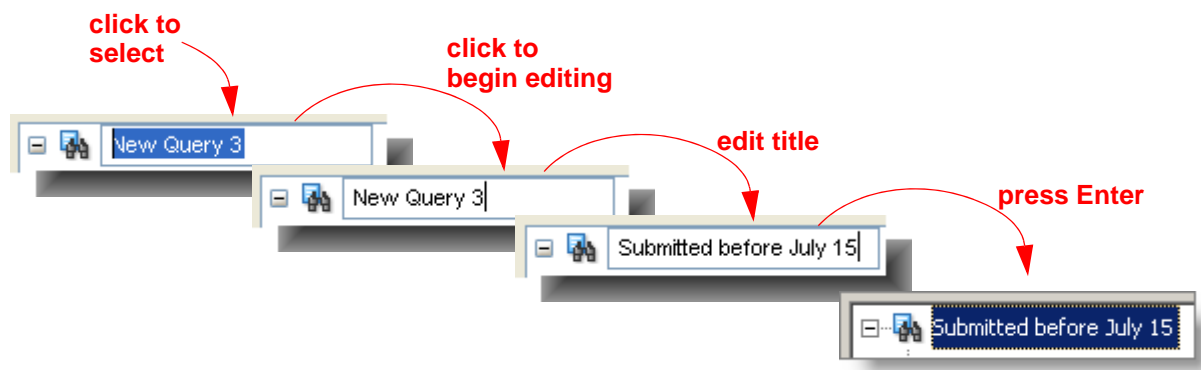
The hierarchy can get deep, because each of the clauses *cls-a*, *cls-b*, etc. can be either simple or compound. For example, the sophisticated query pictured above can be expressed as:

“Retrieve each bug report that is assigned to either **derek**, **jjp**, or **mary**, and applies to product **ColorStar**, and is numbered above 4500.”

The following sections fill in the details of working in the Edit Query window.

### Naming a New Query / Renaming an Existing Query

When you click the  **New Query** button to create a new, empty query, AccuWork assigns it a placeholder title (“New Query *nnn*”). You can edit the title now or whenever the Edit Query window is open: click the title once to select it (if it is not already selected); then click a second time to begin editing it. Not too fast! Double-clicking a title is equivalent to using the expand/collapse control.

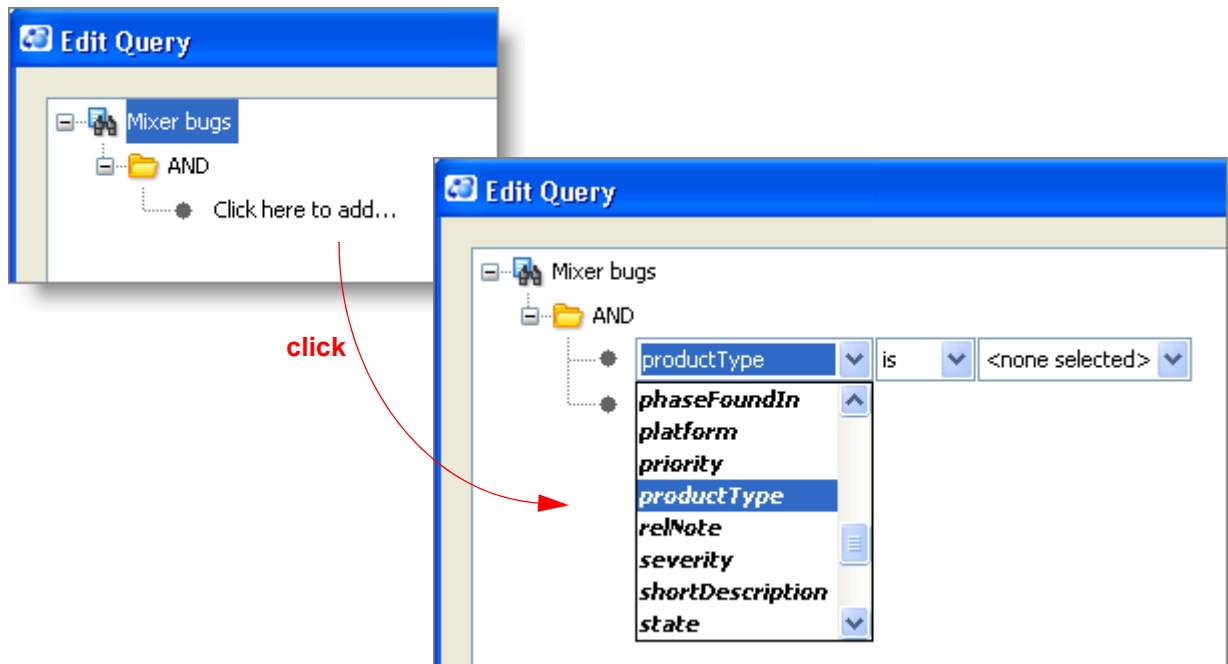


### Creating a Simple Clause

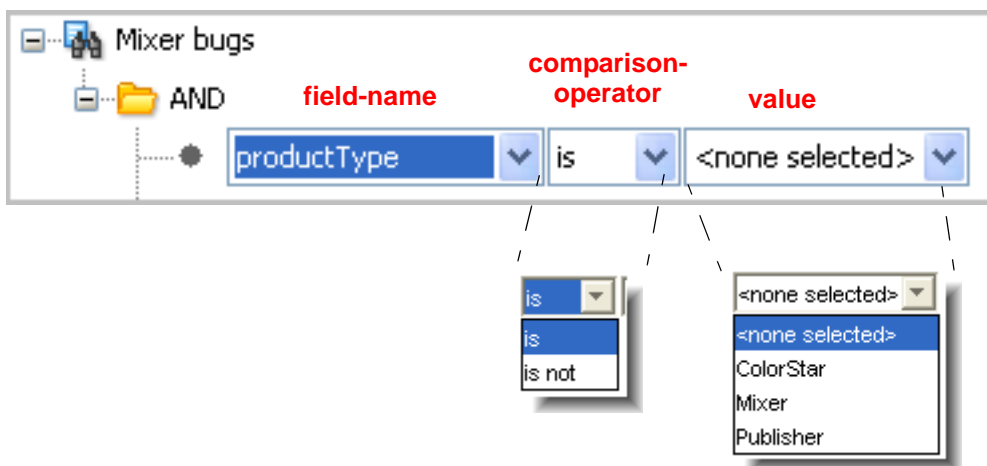
Every query consists at least one simple clause. A simple clause has three parts:

*<field-name>*    *<comparison-operator>*    *<value>*

The point-and-click interface makes creating a simple clause easy and (almost) foolproof. Start by clicking one of the “Click here to add ...” placeholders in the query:



First, you must select the *<field-name>* part of the clause from the list-box containing all the field-names. When you select a field-name, the query editor automatically adjusts the *<comparison-operator>* and *<value>* parts of the clause, based on the selected field. In the example below, the user has selected field-name **productType**, whose value must be one of these names: **ColorStar**, **Mixer**, **Publisher**.



The table below shows all the AccuWork data types, along with the corresponding choices for the *<comparison-operator>* and *<value>* parts of a simple clause. For more on data types, see [Data Types](#) on page 193.

Field Type	Comparison Operators	Values
Text	contains matches does not contain does not match equal to not equal to less than less than or equal to greater than greater than or equal to	Any character string. (Do not enclose it in quotes.) The value is always interpreted as a string literal; there is no way to specify the value of some other field here.  The comparison is always a string comparison, never a numeric comparison. For example, the value <b>3</b> is greater than the value <b>25</b> .  The <b>matches</b> and <b>contains</b> operators — and their negations — are case-insensitive.
Timespan	equal to not equal to less than less than or equal to greater than greater than or equal to	A numeric value, representing an amount of time.  Note: users specify the value in the edit form as a number of <i>hours</i> (e.g. 7.5); an XML-format dump of the issue record created by the <b>Export</b> command reports the value as a number of <i>minutes</i> (e.g. 450).
Choose	is is not	One of the strings specified in the definition of this field in the Schema Editor.
List	is is not	One of the strings specified in the definition of a particular named list in the Schema Editor.
User	is is not is member of is not member of	One of the principal-names in the user registry maintained by the AccuRev server. Alternatively, a user-group defined in the registry.
Timestamp	is is not is before is after is before or equal to is after or equal to	An AccuRev timestamp.
Attachments	contains	Any character string. This string is compared to the <b>Name</b> of each of an issue record's attachments. See <a href="#">Creating Attachments to an Issue Record</a> on page 169.
internal	equal to not equal to less than less than or equal to greater than greater than or equal to	An integer, identifying a particular AccuWork issue record ( <b>issueNum</b> field) or a particular AccuRev transaction ( <b>transNum</b> field).

As you “fill in the blanks” to create simple clauses, you’ll notice that AccuWork allocates new “Click here to add ...” placeholders. It makes sure that one of these placeholders is always available at each level of the query.

### Creating a Compound Clause

A compound clause combines any number of subclauses together, using the same logical operator: AND or OR. (The NOT operator is not supported.) The subclauses to be combined can, themselves, be either simple or compound:

*simple AND simple*  
*simple AND compound*  
*simple OR compound OR simple*  
*compound AND compound AND simple AND compound AND compound*  
etc.

Note: a compound clause can contain a single subclause. This is logically equivalent to using the subclause by itself. In fact, the most basic query you can create is a compound AND clause (or compound OR clause) with one simple subclause.

What about this?

*simple OR simple AND simple*

Well, it depends. This might be a compound OR clause within a compound AND clause:

*( (simple OR simple) AND simple )*

Or it might be a compound AND clause within a compound OR clause:

*( simple OR (simple AND simple) )*

Whichever meaning was intended, AccuWork models the logical statement as one compound clause nested within another.

We saw above that the query editor automatically creates placeholders for simple clauses. But you must explicitly insert a compound-clause placeholder yourself, then fill in the subclauses. We’ll demonstrate with an example:

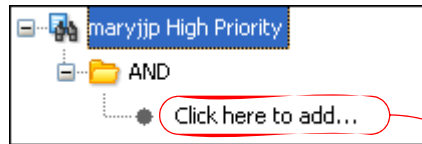
“Retrieve all **high**-priority bug reports assigned to either **mary** or **jjp**”

... which becomes a compound OR clause within a compound AND clause:

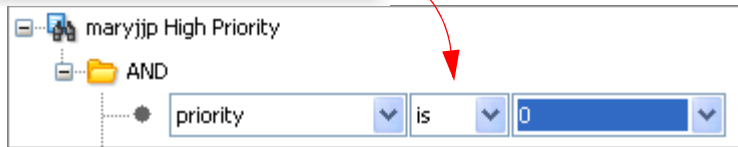
*( fix\_priority is **high** AND (assign\_user is **mary** OR assign\_user is **jjp**) )*

Here’s the procedure for creating this query:

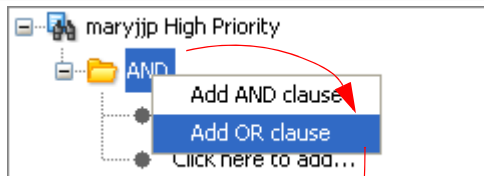
**1. create a new query and name it**



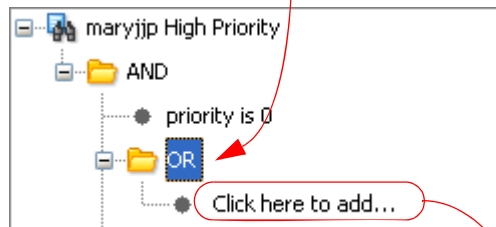
**2. click the placeholder and fill in a simple clause**



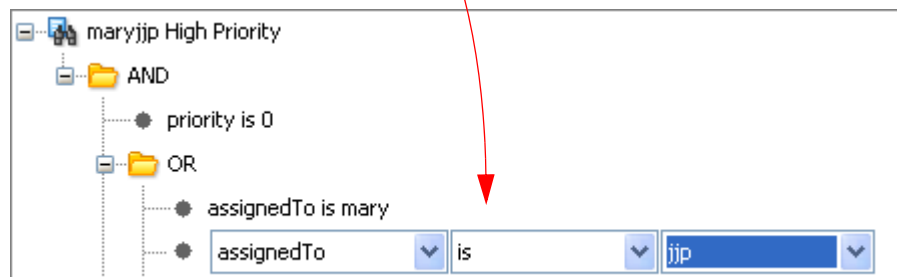
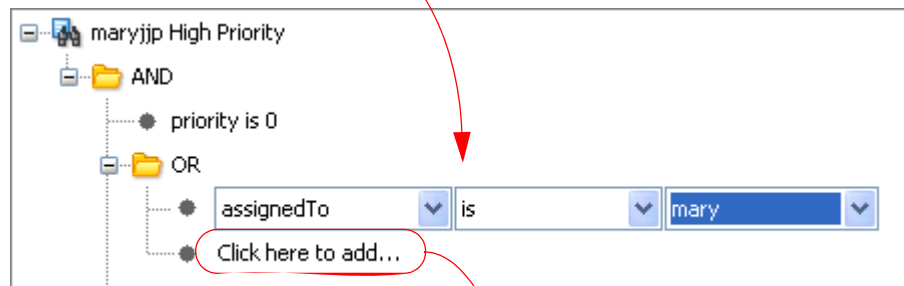
**3. right-click at the parent level and create an OR subclause**



**4. click the placeholder and fill in a simple clause**

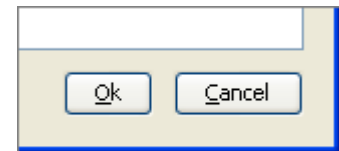


**5. click the new placeholder and fill in another simple clause**



## Closing the Edit Query Window

Whether you're composing a new query or revising an existing one, you end by saving your work (**Ok** button) or discarding it (**Cancel** button). AccuWork automatically executes the query and displays a results table in the Query Results pane.



At this point, you'll often want to work on the design of the results table: specify additional fields to be displayed, and adjust the widths and order of the columns. For more on this, see *Working with a Results Table* on page 185.

## Working with a Results Table

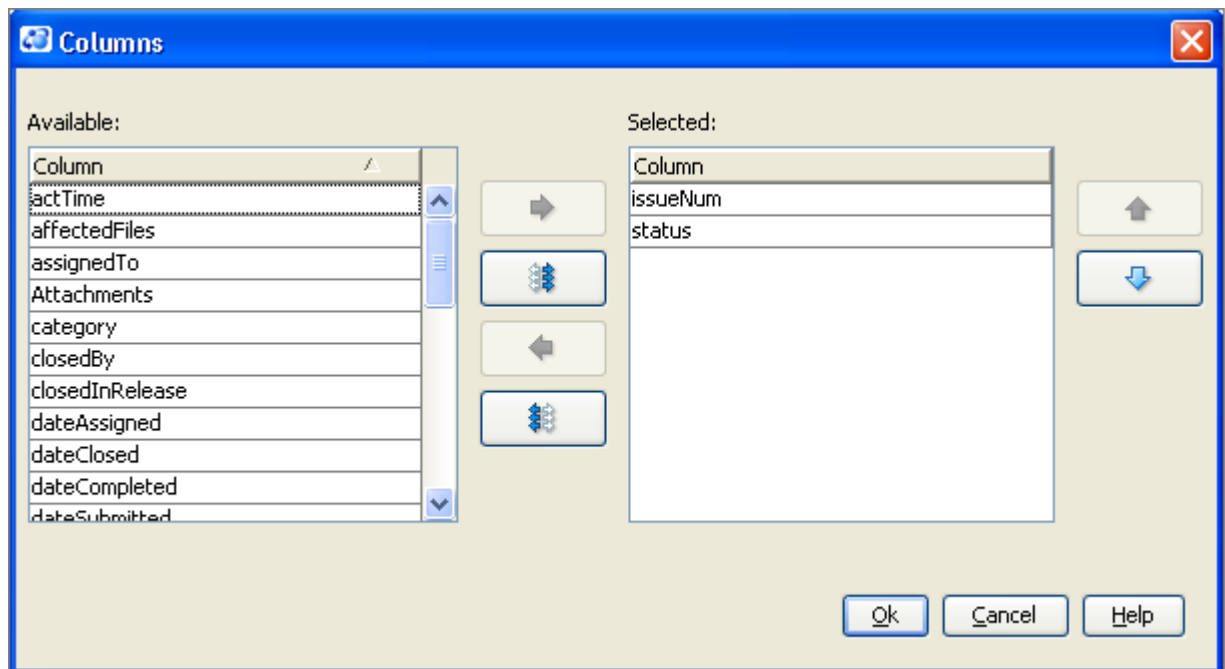
Each query has its own results-table design: a set of columns (fields), in a particular order and with particular column-widths. The results table's design can also include a single-level or multiple-level sort order for the rows (issue records).

When you create a new query, AccuWork automatically starts off the results table with a couple of columns, including **Issue** (issue-number). You can add more columns at any time. You can also remove the **Issue** column from the table.

## Selecting Columns to Appear in the Results Table



Use the **Columns** button in the Query Results toolbar to launch a window in which you select the columns to appear in the results table, and their order. (You can also change the column order in a results table using drag-and-drop — see below.)









Note that:



- The names that you work with in the Columns window are the official field-names defined in the issues database schema.
- The names that appear as column headers in the results table are the labels that appear on the database's edit form.

For some queries, you may want to include just a few important columns in the results table; for others (e.g. a complete data-dump), you may want to include all the fields defined in the issues database. Working in the Columns window, you can:

- Select a column label and use the  and  buttons to move it between the **Selected** (appears in results table) and **Available** (does not appear in results table) lists.
- Select a column label in the **Available** list and use the  and  buttons to change the order of the columns.
- Use the  and  buttons to move all columns labels to the **Available** or **Selected** list.

When you're done, click **Ok** to apply the changes you've made, or click **Cancel** to discard the changes. You can further adjust the columns of a query results table using the techniques that work with all AccuRev GUI tables. See *Working with Tables* on page 4 of the *AccuRev User's Guide (GUI Edition)*.

All the changes you make, both in the Columns window and in the Query Results pane, are preserved when you invoke the **Save All** command.

It's likely that you won't be able to see the complete text strings entered into multiple-line text fields, no matter how wide you've made the results-table column. This limitation doesn't apply when you "print" the results table to an HTML file: the complete contents of each field is included in the HTML table. (See *Printing Query Results* on page 187.)

If you change the column layout in the Query Results pane after running a query, you don't need to rerun it — even if you add or delete columns. AccuWork automatically updates the results table in this case.

## Working with the Records Listed in a Results Table

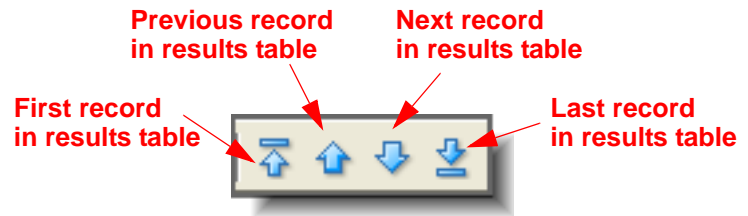
In many cases, browsing the results table produced by running a query may be all you need to do. But in other cases, you may want to see a selected issue record in the context of its edit form:


- The results table is read-only. To modify an issue record, you must access it through its edit form.
- That data you need may be in a field that is not included in the results table. Instead of adding a column to the results table, you can use the edit form to display all the fields.

Use either of the following techniques to retrieve the complete records listed in the results table:

- **Full-size edit form in a new tab** — select a row of the query results table and click the **Open Issue** button in the Query Results toolbar. (Alternatives: right-click the results-table row and select **Open Issue** from the context menu; or double-click the row in the results table.)

This opens a new tab containing  
When you open an issue record from a results table, browse arrows are enabled in the edit form's toolbar. This makes it easy to view, in the edit form, some or all of the records selected by a query.



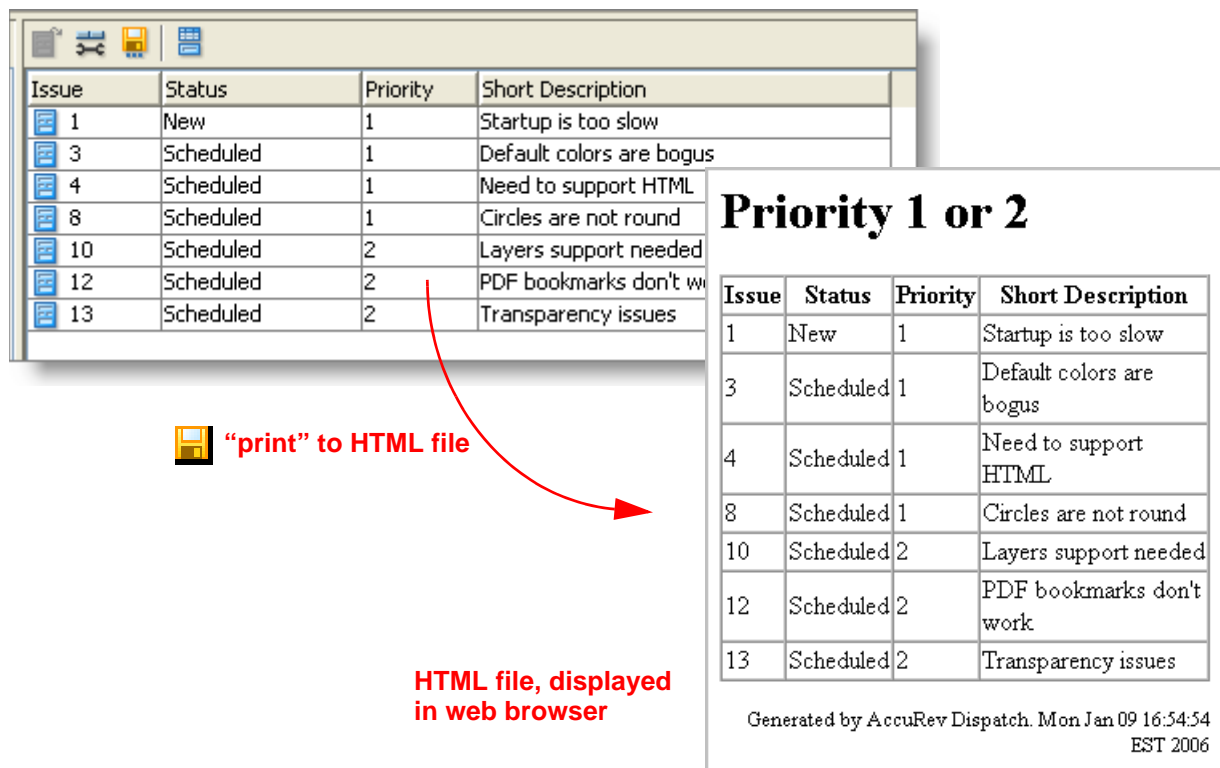
- **Reduced-size edit form pane in the Queries tab** — Click the  **Show Issue Form** button in the Query Results pane toolbar. The edit form that appears is fully functional. As you select records in the query results table, they are automatically loaded into the edit form.

If you wish, modify the record as described in [Viewing and Modifying an Existing Issue Record](#) on page 172. After modifying a record, you may want to rerun the query in order to update the results table; this doesn't occur automatically.

## Printing Query Results



Use the **Export Table** button on the Queries toolbar to save the current contents of the results table to an HTML-format file. The file contains an HTML table; each cell changes height and width when you view it with a Web browser and adjust the size of the browser window. This is particularly useful for viewing the contents of multiple-line text fields.



**Priority 1 or 2**

Issue	Status	Priority	Short Description
1	New	1	Startup is too slow
3	Scheduled	1	Default colors are bogus
4	Scheduled	1	Need to support HTML
8	Scheduled	1	Circles are not round
10	Scheduled	2	Layers support needed
12	Scheduled	2	PDF bookmarks don't work
13	Scheduled	2	Transparency issues

Generated by AccuRev Dispatch. Mon Jan 09 16:54:54 EST 2006



# Designing Issues Databases and Edit Forms: The Schema Editor

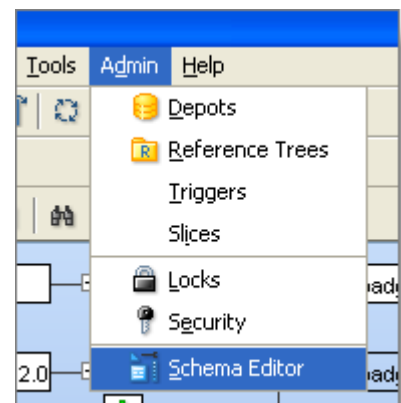
This chapter describes how to use the AccuWork Schema Editor to define a AccuWork issues database, containing a collection of issue records. Each depot can contain one issues database, along with a custom-designed edit form, through which users create and modify the issue records. You can make the edit form “smart” by defining validations (edit checks) that specify default values, required fields, and interrelationships among multiple fields.

## Invoking the Schema Editor

You perform all AccuWork database-design work on the Schema Editor tab, which you display using the **Admin > Schema Editor** command.

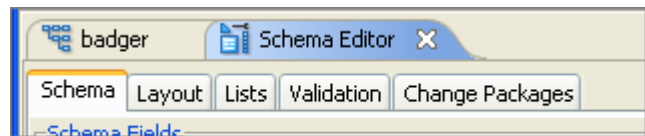
The first time you invoke this command in a particular depot, AccuWork offers to use the repository’s default schema. Accepting this offer copies a set of XML-format configuration files from subdirectory **dispatch/config** of the **site\_slice** directory to this depot.

Note: the default schema does not actually become the schema for this depot until you click the Schema Editor’s **Save** button. See *Defining Database Fields* below.



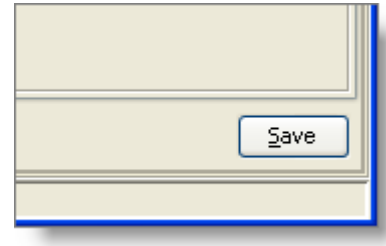
The Schema Editor tab includes these subtabs:

- You define the AccuWork database’s fields on the **Schema** subtab.
- You design the database’s edit form on the **Layout** subtab.
- You define multiple-choice lists, each of which can constrain the values of one or more fields, on the **Lists** subtab.
- You define field validations, or edit checks, on the **Validation** subtab.
- You monitor and control certain aspects of the integration between AccuRev and AccuWork on the **Change Packages** subtab.



## Saving Changes to the Schema

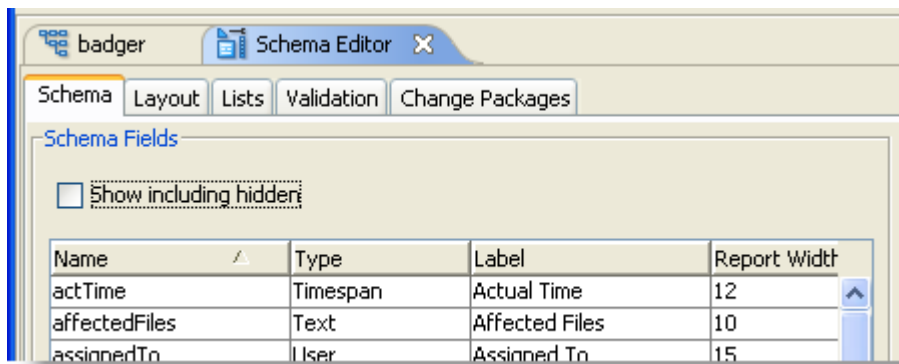
At any time while working in the Schema Editor, you can click the **Save** button in the lower right corner of the Schema Editor tab. This saves the current state of the schema to a set of XML-format files in subdirectory **dispatch/config** of the depot directory (slice) in the AccuRev repository:



- Contents of the **Schema** subtab: **schema.xml**
- Contents of the **Layout** subtab: **layout.xml**
- Contents of the **Lists** subtab: **lists.xml**
- Contents of the **Validation** subtab: **logic.xml**
- Contents of the Change Package Results section of the **Change Packages** subtab: **cpk\_fields.xml**
- Contents of the Change Package Triggers section of the **Change Packages** subtab: **cpk\_promote\_queries.xml**

## Defining Database Fields

To begin designing an issues database, open a Schema Editor tab and go to the Schema subtab. (This happens automatically if you accept AccuWork's offer to use the repository's default schema.)



If you are not using the default schema, AccuWork initializes the Schema Fields table with two fields.

- **issueNum**: An integer-valued field that records the position of the issue record in the depot's issues database. This number is assigned when a user creates the record (i.e. at the first **Save** on the edit form), and it never changes.
- **transNum**: An integer-valued field that records the transaction number of the most recent update to the issue record. The transaction resides in the depot's transaction log — the same log that records AccuRev transactions, such as **keep** and **promote**. The type of the issue-record-update transaction is **dispatch**.

Note that a record's **issueNum** value never changes, but its **transNum** value changes each time a user **Saves** the record. The integration between AccuRev and AccuWork also updates issue records, and so changes the value of the **transNum** field. Also note that **issueNum** and **transNum** are indexes into two different databases.

## Adding and Removing Fields from the Database Schema

You can define any number of additional fields in the issues database schema. Follow these steps for each new field:

1. Click the **Add** button at the bottom of the Schema subtab.

2. Fill in the **Create New Field** window that appears:

- **Name:** The official field-name of the field. Users of the AccuWork database don't ever need to know this name — they know the field by its label. The field-name must not contain any SPACE characters. Validations must be expressed in terms of the field's name, not its label.
- **Type:** One of the data types supported by AccuWork. See *Data Types* below.
- **Label:** The field-label character string that identifies the field on the database's edit form. A field-label can contain SPACE characters (e.g. **Last Name**).
- **Report Width:** An integer that determines the relative width of the field in the HTML table created when the user clicks **Export** on the edit form of an individual issue record.

3. Click **OK** to close the **Create New Field** window.

4. In the Field Values box to the right of the Schema Fields table, specify additional information about the field. The kind of information required varies with the data type (see *Data Types* below).

issueNum	internal	Issue	10
problem_description	Text	Problem	30
problem_headline	Text	Summary	30
program_name	List	Program	15
program_release	List	Release	6
state	Choose	State	10
submit_date	Timesta...	Submitted on	15
submit_user	User	Submitted by	10

Height

Width

Repeat the steps above as often as required to create new fields in the AccuWork database.

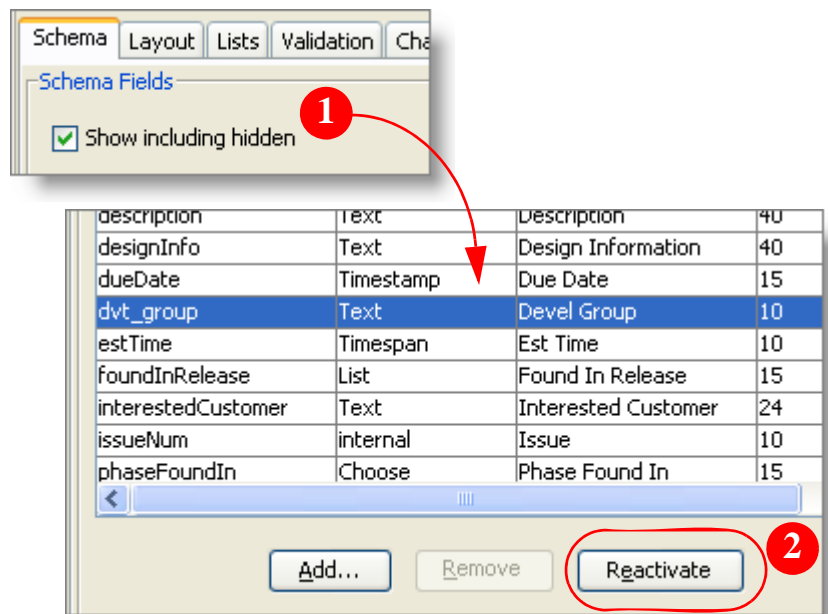
Note: your field definitions are not saved until you click the **Save** button in the lower-right corner of the Schema Editor tab. You cannot save your work until you place at least one field in the database’s edit form (Layout subtab).

To remove an existing field (except for **issueNum** and **transNum**, as noted above), select it and click the **Remove** button. The field disappears from the Schema tab and can no longer be used in the edit form (see *Designing an Edit Form* on page 194). But any data stored in existing issue records is preserved.

Note: when you remove a field from the schema., AccuWork automatically checks whether the field is used in the issue database’s edit form. If so, it removes the field from the edit form

You can restore a removed field to the database schema:

1. Check the **Show Including Hidden** checkbox. All removed fields appear in the list, with a gray background.
2. Select the field to be restored, and click the **Reactivate** button.



## Integrating Configuration Management and Issue Management: the ‘affectedFiles’ Field and Change Packages

If you wish to enable the integration of a depot’s version-controlled files and its AccuWork issue records, define a database field whose name is **affectedFiles**. The field’s type must be “text”; see *Data Types* below. You can choose any label for the field. (Such a definition is included in the default AccuWork database schema.)

The integration also depends on the enabling of a built-in AccuRev trigger procedure:

```
accurev mktrig -p <depot-name> pre-promote-trig client_dispatch_promote
```

The integration routine writes the transaction number of each **promote** command to the **affectedFiles** field of a particular issue record. Alternatively, in the AccuRev Enterprise version of AccuWork, the integration routine records each promoted version in the issue record, in a special section named **Changes**. This section is maintained automatically by AccuWork — you don’t need to define any database fields to enable this additional aspect of the integration.

For details, see *Integrations with AccuWork* on page 154 of the *AccuRev User's Manual (CLI)*, and *Change Packages and Integrations between Configuration Management and Issue Management* on page 209.

## Data Types

Each field you define in the **Create New Field** window must have one of the AccuWork data types listed in the table below.

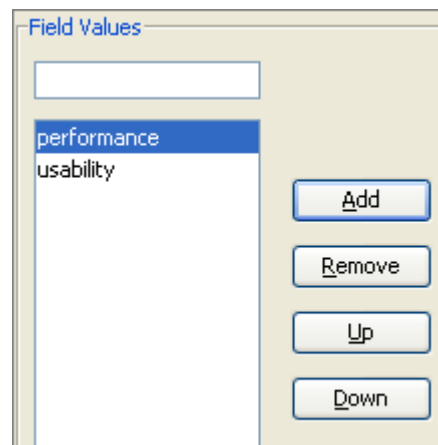
Field Type	Possible Values	Required Additional Information in “Field Values” Box
text	Any character string. For multiple-line fields, the string can include line-terminators.	<p><b>Height:</b> number of lines displayed for the field in the edit form (default: 1). For multiple-line fields (height &gt; 1), the edit form includes an expand/contract button to increase/decrease the number of lines displayed.</p> <p><b>Width:</b> relative width of field in the edit form.</p>
choose	One of the strings specified in the Field Values box for this field.	<p>A set of strings.</p> <p>In the edit form, a list-box containing this set of strings is offered to the user.</p>
timestamp	An AccuRev timestamp.	<p>Granularity (year, month, day, hour, minute, or second).</p> <p>In the edit form, the user fills in fields that indicate a time, to the granularity you specify here.</p>
user	One of the principal-names in the user registry maintained by the AccuRev server.	<p>None.</p> <p>In the edit form, a list-box containing all principal-names is offered to the user.</p>
timespan	A numeric value, indicating a number of hours. Decimal values (e.g. 4.5) are allowed.	<p>None.</p>
list	One of the strings specified in the definition of a particular named list.	<p>The name of an existing list, defined on the <b>Lists</b> subtab. (See <i>Defining Multiple-Choice Lists</i> below.)</p> <p>In the edit form, a list-box containing the set of strings defined in the named list is offered to the user.</p>
log	Any character string (variant of text field type)	<p>In the edit form, an Add Text control appears above the input field. The user can type directly into the input field, or can click the Add Text control and create a “log entry” in the popup window that appears.</p>
attachments	A set of attachment definitions.	<p><b>Height, Width:</b> the height (number of lines) and width (approximate number of characters) of the edit-form field that lists the attachment definitions.</p>
internal	A positive integer.	<p>None.</p> <p>You cannot create a field with this data type; it is used only by the built-in fields <b>issueNum</b> and <b>transNum</b>.</p>



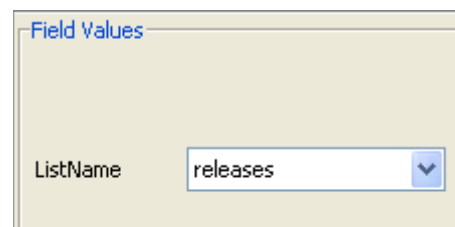
## Defining Multiple-Choice Lists

The **choose** and **list** data types are similar:

- For a **choose** field, the set of possible values — an ordered list of strings — is part of the individual field definition. You enter the possible-values list in the Field Values box for that field.
- For a **list** field, the set of possible values is also an ordered list, but it is *not* part of the individual field definition. In the Field Values box, you specify the name of one of the lists created on the Lists subtab. Any number of **list** fields in a AccuWork database can use the same named list:



The 'Field Values' dialog box for a 'choose' field. It features a text input field at the top, a list box containing 'performance' and 'usability' (with 'performance' selected), and four buttons on the right: 'Add', 'Remove', 'Up', and 'Down'.

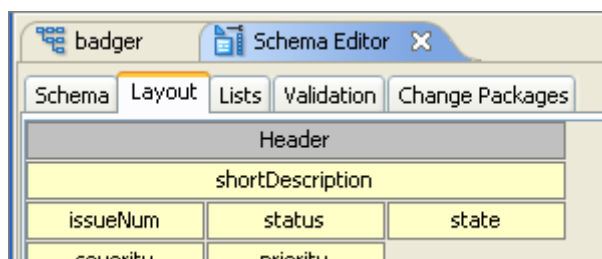


The 'Field Values' dialog box for a 'list' field. It shows a 'ListName' label and a dropdown menu with 'releases' selected.

The mechanics of defining the ordered list is similar in the “local” case (for an individual **choose** field) and in the “global” case (on the Lists subtab, for use by any number of **list** fields). On the Lists subtab, you must supply a ListName for the list; for an individual **choose** field, the possible-values list doesn’t need or have a name.

## Designing an Edit Form

You design the edit form for a AccuWork database on the Layout subtab.



The 'Schema Editor' window with the 'Layout' subtab selected. It shows a table layout for an edit form. The table has a header row and several data rows. The fields are arranged in columns: 'shortDescription', 'issueNum', 'status', 'state', 'severity', and 'priority'.

Header					
shortDescription					
issueNum	status		state		
severity	priority				

An edit form consists of field-labels and corresponding edit-widgets, arranged in rows and columns. The field-labels come from the Labels column of the Schema Fields table (Schema subtab). The edit-widget for a field depends on its data type and, in some cases, on the additional Field Value information.

	column	column
row	Assigned To <input type="text" value="&lt;none selecte..."/>	Date Assigned <input type="text" value="yyyy mm dd hh mm ss"/> 2004 5 25 19 38 51
row	Target Release <input type="text" value="&lt;none selected&gt;"/>	Due Date <input type="text" value="yyyy mm dd"/>
row	Date Completed <input type="text" value="yyyy mm dd"/>	
row	Est Time <input type="text"/>	Actual Time <input type="text"/>
row	<div> <div> </div> <div> The startup routine is so slow that many users just give up. </div> </div>	

field-label edit-widget

"Description" field spans two columns

The Layout subtab provides a drag-and-drop canvas on which you design the edit form's rows and columns. On this canvas, each field-label/edit-widget pair is represented by a yellow box. The following screen shot shows the layout that defines the edit form illustrated above:

each "table" defines a separate tabbed page on edit form

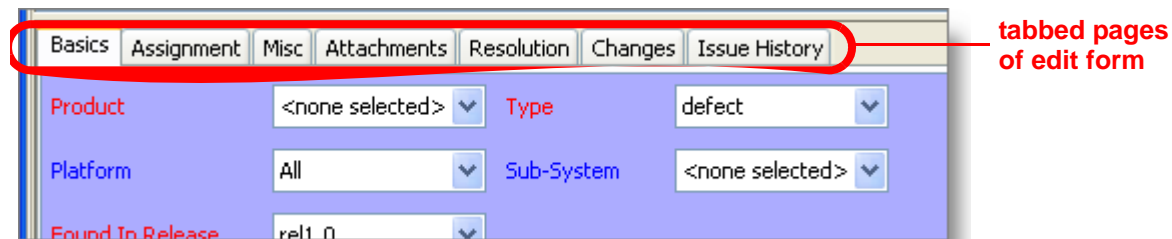
this field spans multiple columns

field-names used, not field-labels

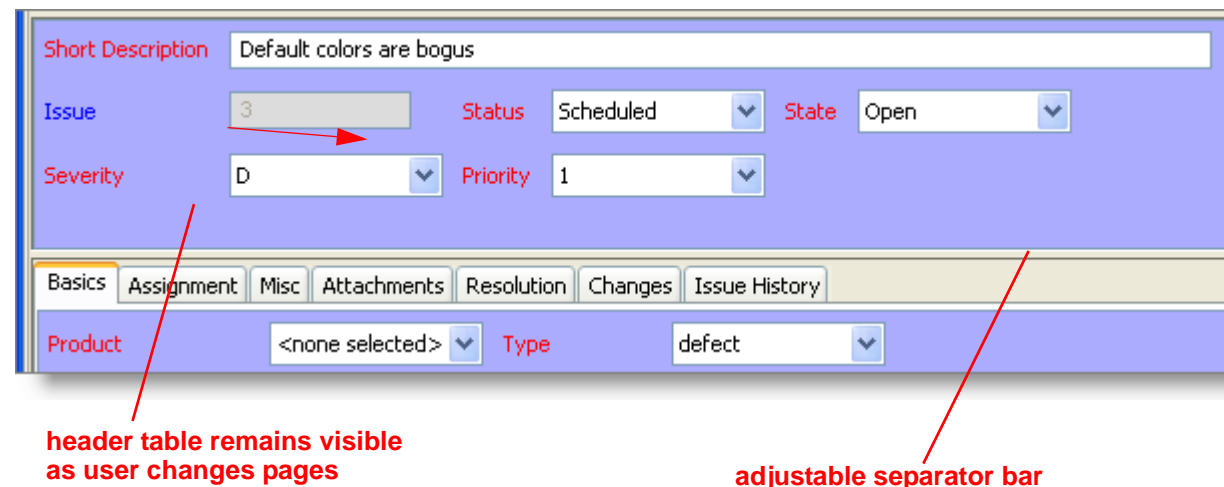
field-names used, not field-labels

As the screen-shot annotations indicate, the yellow box contains a field-name, not a field-label. On the edit form, the field-label and the corresponding edit-widget will appear, side by side, at this position. You can also expand a yellow box to make it span two or more columns.

You can organize the fields into multiple “tables”, each of which defines a separate tabbed page in the edit form.



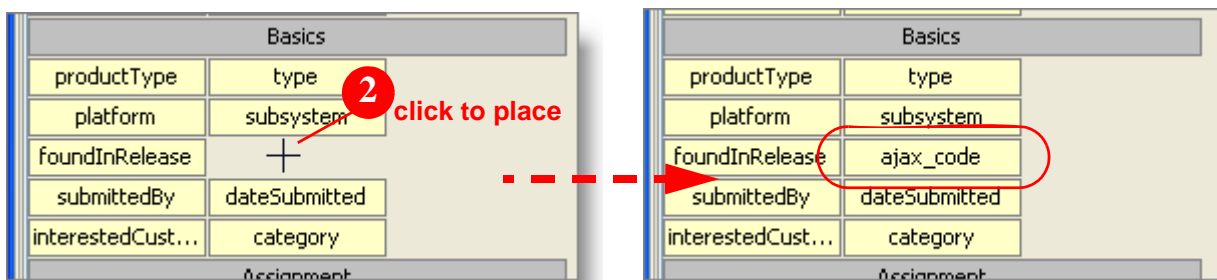
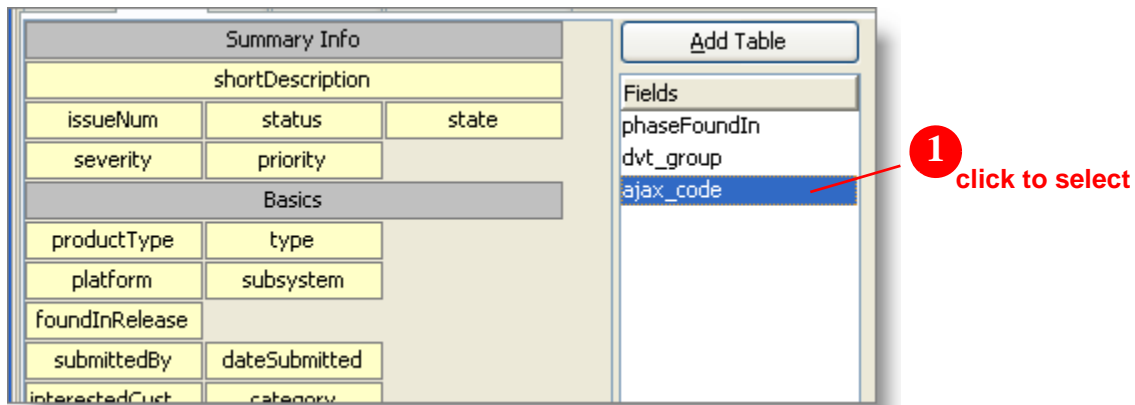
One of the tables can be a header table. Instead of defining a separate tabbed page, a header table defines a set of fields that always appear on the edit form, no matter which tabbed page is currently visible.



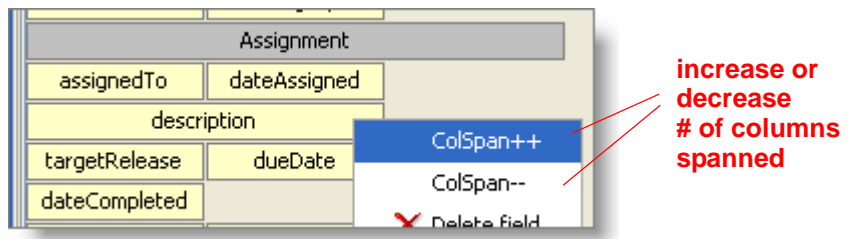
## Form Layout Operations

This section describes how to perform the various edit-form design operations on the Layout subtab.

- **Adding a field to the edit form:** The **Fields** list-box offers all fields that do not currently appear in the edit form. Click to select a field in this list-box. Then, click at the empty location on the canvas where you want to place the field.



- **Arranging fields into rows and columns:** Drag-and-drop the yellow boxes to the desired location on the canvas. Drop a box on top of another one to push it rightward or downward.
- **Changing column-spanning:** Right-click a yellow box, and select one of the spanning operations from the context menu.

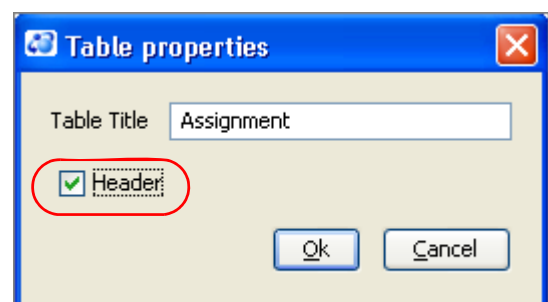


- **Creating a new page in the edit form:** Click the **Add Table** button above the **Fields** list-box. Fill in the Table Title field in the dialog box that appears, then click **OK**. This title appears on the page's tab in the edit form.

A gray box with the specified title appears at the bottom of the layout. Fields that you place below this box will appear on the new page of the edit form.

- **Creating a header table for the edit form:** When creating a new page for the edit form (see above), check the **Header** checkbox if you want it to be the header table. (If another table is currently selected as the header table, it is automatically deselected.)

You can also redefine an existing page to be a header table: right-click the gray box and select



**Properties.** All fields in the header table will appear at the top of the database's edit form, above every tabbed page.

Remember that an edit form can have at most one header table. Its fields always appear at the top of the edit form, even if it is not the topmost table on the Layout tab.

- **Renaming a page:** Right-click on the gray box, select **Properties**, and change the entry in the Table Title box.
- **Arranging fields into multiple pages:** Drag-and-drop the yellow boxes (fields) and the gray boxes (pages). Fields that you place below a gray box will appear on the corresponding page of the edit form.
- **Removing a field or page:** Right-click the yellow box (field) or gray box (page), and select **Delete** from the context menu. Deleting a page doesn't delete the fields currently on the page — it just deletes the gray box, effectively merging the fields into the page above.

Note: you cannot delete the first gray box; an edit form must have at least one page!

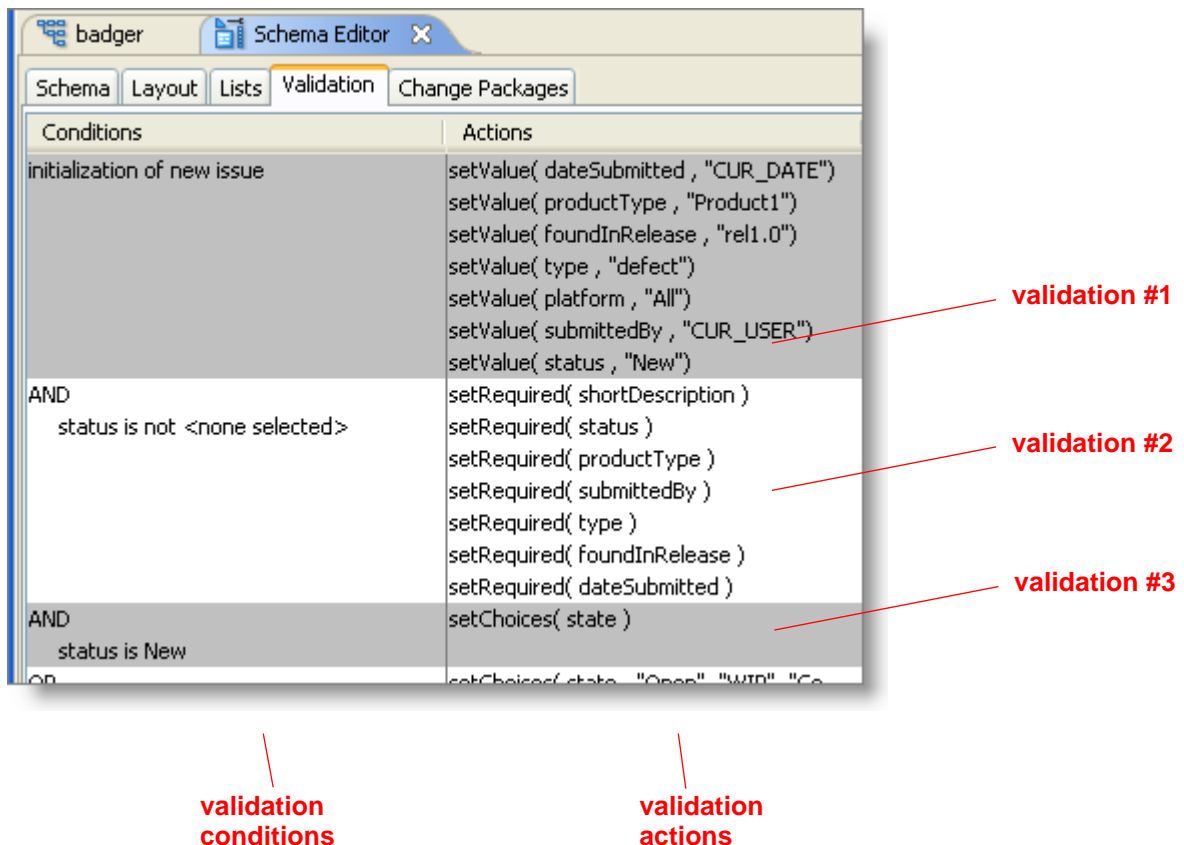
- **Setting the edit form's colors:** Click one of the colored buttons at the right side of the Layout subtab. Then select a color from the **Pick a color** window. You can set these colors:
  - **Foreground:** the color of the edit form's field-labels.
  - **Required Foreground:** the field-label color for fields where a value must be specified. Required fields are defined on the Validations subtab. See *Defining Edit Form Validations* on page 198.
  - **Background:** the color of the edit form's background.

Note: The "Foreground" and "Background" settings are currently not used.

## Defining Edit Form Validations

Users of a AccuWork database create and modify issue records through the database's edit form. To increase the efficiency and accuracy of this process, you can create a set of validations to be applied as the user works with the edit form. (Validations are sometimes called "edit checks".)

You create and maintain the set of validations using a point-and-click interface. AccuWork displays the current validations in tabular format; each one takes the form "if a certain condition is true, then perform a particular set of actions".



The following kinds of validations are available:

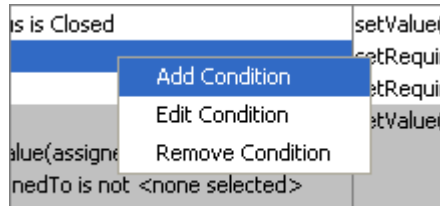
- Initialize a field in a new issue record with a specified value.
- Require a value to be entered in a field.
- Based on a condition, make one or more of these changes:
  - Set a **text** field to a specified value.
  - Set a **user** field to a particular value — for example, to the current user.
  - Set a **timestamp** field to a particular value — for example, to the current time.
  - Make one or more fields into required fields.
  - Change the possible-values list for a **choose** field.
  - Set the entire issue record, a particular page (tab) of the issue record, or a particular field to be read-only.

The condition you specify is essentially the same as a AccuWork query; if the condition is true for (“selects”) the current record, the specified changes are applied to the edit form.

## Initializing Field Values in a New Issue Record

The first entry in the table of validations is special. Its condition is always “initialization of new issue”, so that its actions are performed exactly once: between the time the user invokes the **New Issue** command and the time the new edit form appears.

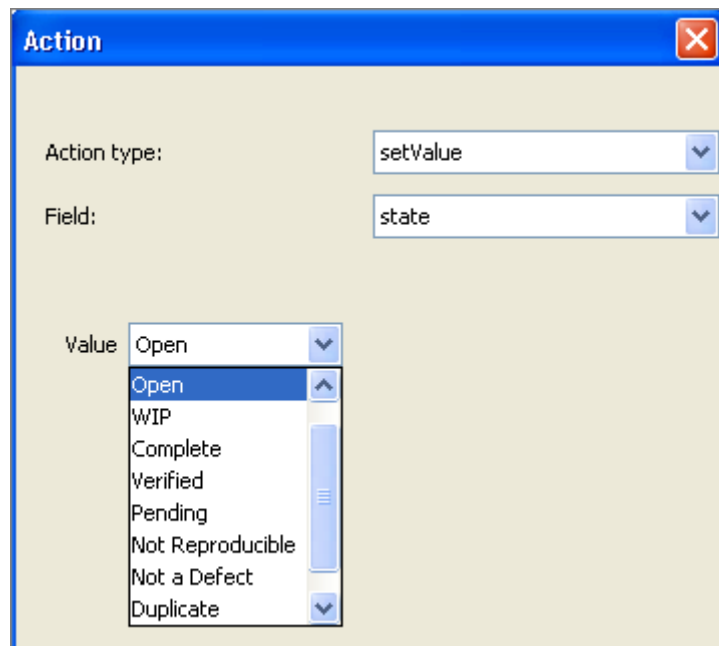
To define an action, right-click in the **Actions** column, then select **Add Action** from the context menu.



An **Action** window appears, in which you define the action. For initialization of a new issue record, the only appropriate action type is **setValue**.

Note: some releases of AccuWork allow you to include **setRequired** and **setChoices** actions in a record initialization. Such actions will be ignored when a new record is initialized.

After setting the action type to **setValue**, select a field to be initialized and specify the initial value. The **Value** edit-widget adjusts to the selected field. In this example, **state** is a “choose” field, so the list-box offers the field’s predefined choices as the initial value.



## Conditional Validations

The setting of initial field values is an unconditional validation: it happens every time a **New Issue** command is invoked. All other validations are conditional: a certain set of actions are performed if, and only if, a certain condition is met.

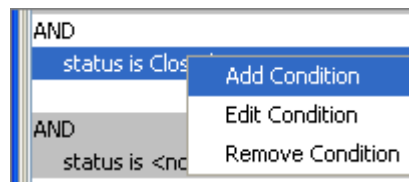
The unconditional setting of initial field values occurs just once; but the conditional validations are performed repeatedly: when an edit form first appears *and* each time the user changes any field value. Each repeat involves:

- Clearing the “required” status of all fields.

- Performing all validations (except for that first one: “initialization of new issue”). If a validation’s condition is true, the corresponding actions are invoked.

## Specifying the Condition

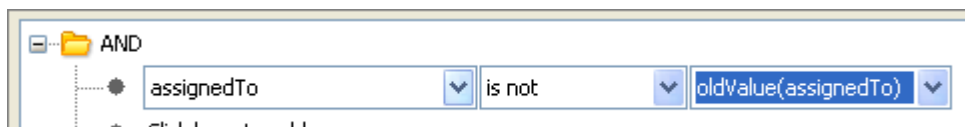
To create a new conditional validation — that is, a new shaded or unshaded row in the table — right-click anywhere in the Conditions column, then select **Add Condition** from the context menu. Then proceed to construct the “if” clause, as described above.



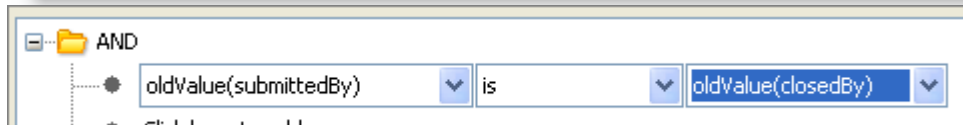
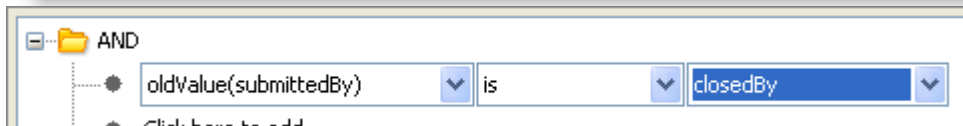
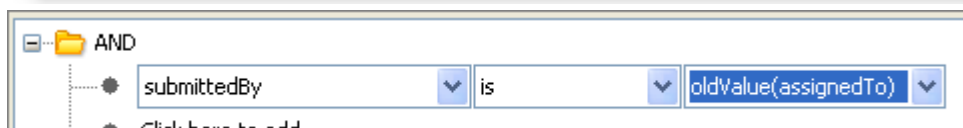
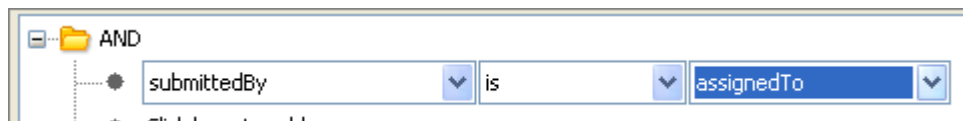
The same context menu provides commands for revising the “if” clause of an existing conditional validation, and for removing a conditional validation.

Specifying the condition itself — the “if clause” — of a conditional validation is very much like specifying a AccuWork database query. (Queries are described in section [Using AccuWork Queries](#) on page 174.) But there are some differences: in a validation condition, you can:

- Compare the current value of a field with its “old” value — that is, compare the value currently displayed for the field vs. the value stored in the AccuWork database by the most recent **Save**.

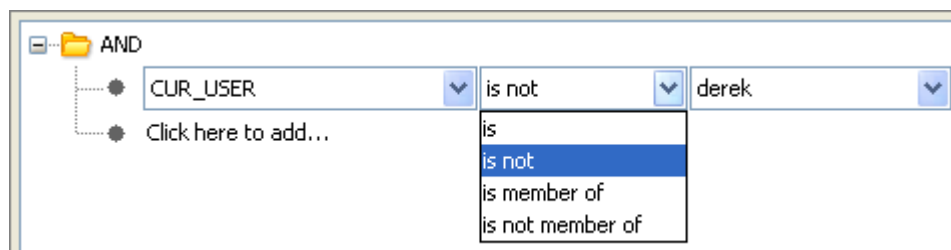


- Compare the (current or old) value of a field with the (current or old) value of another field in the same record.



- Test the AccuRev user identity or group membership of the person using the edit form (CUR\_USER).

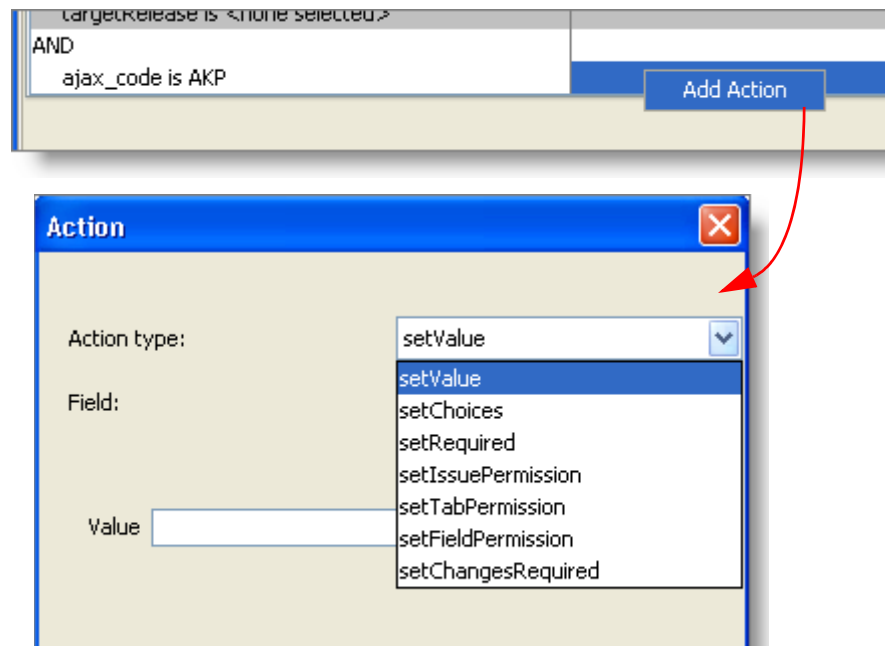




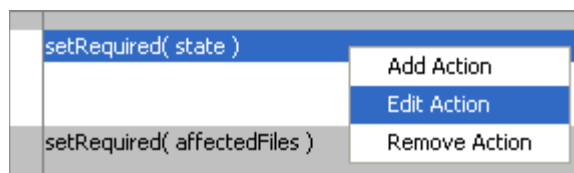
Query conditions cannot make such field-to-field comparisons; they can only compare field values to literal values.

## Specifying the Actions

Each conditional validation can include any number of actions. You create an action by right-clicking in the Actions column of the validation, and selecting **Add Action** from the context menu. This displays a window in which you define the action.



After you've created one or more actions for a conditional validation, you can use the same context menu to revise or delete individual actions.



The following sections describe the actions that you can define to be performed if the validation condition is true:

- *Setting a Field Value* (**setValue**)
- *Revising the Choices for a “choose” Field* (**setChoices**)
- *Requiring a Value to be Entered in a Field* (**setRequired**)
- *Setting Permissions on All or Part of the Issue Record* (**setIssuePermission**, **setTabPermission**, **setFieldPermission**)
- *Requiring Change Set Entries* (**setChangesRequired**)

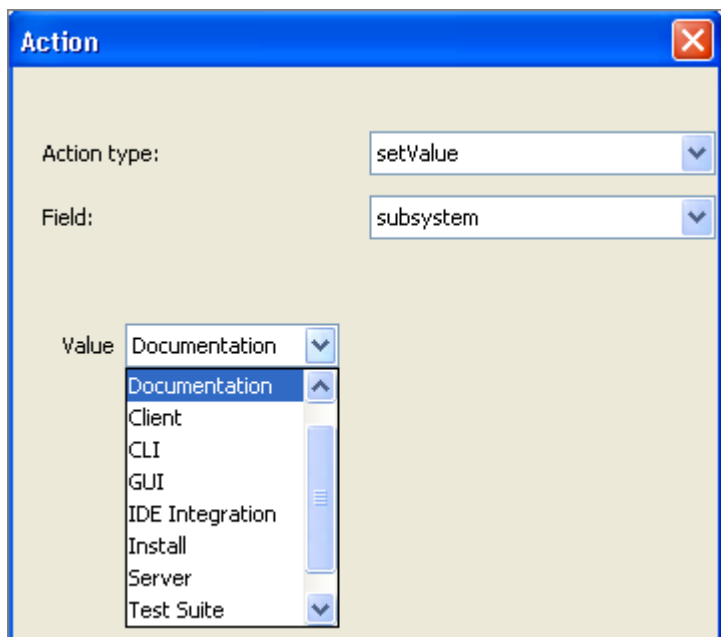
Each validation can invoke any number of actions, of any type.

## Setting a Field Value

In addition to initializing field values (see *Initializing Field Values in a New Issue Record* on page 200), you can set the values of fields while the user is working with the record, based on certain conditions. For example, if the user enters **ColorStar** in the **program\_name** field, a validation could automatically set the **fix\_priority** field to **high**. (Management has mandated rapid improvement in the robustness of the ColorStar application.)

To define a **setValue** action for a validation condition:

1. In the Action window, select **setValue** as the action type, and select the field to be set.
2. The **Value** edit-widget adjusts to the selected field, just as it does when you’re initializing a field in a new issue record.



For a “text”, “log”, or “timespan” field, you enter a text string as the value; for a “date” field, you specify a date in the standard way; for other field types, you use a list-box to specify any of the field’s predefined values.

## Revising the Choices for a “choose” Field

The definition of each field of type “choose” includes an ordered list of strings. The user fills in this field by choosing one of the strings from a list-box.

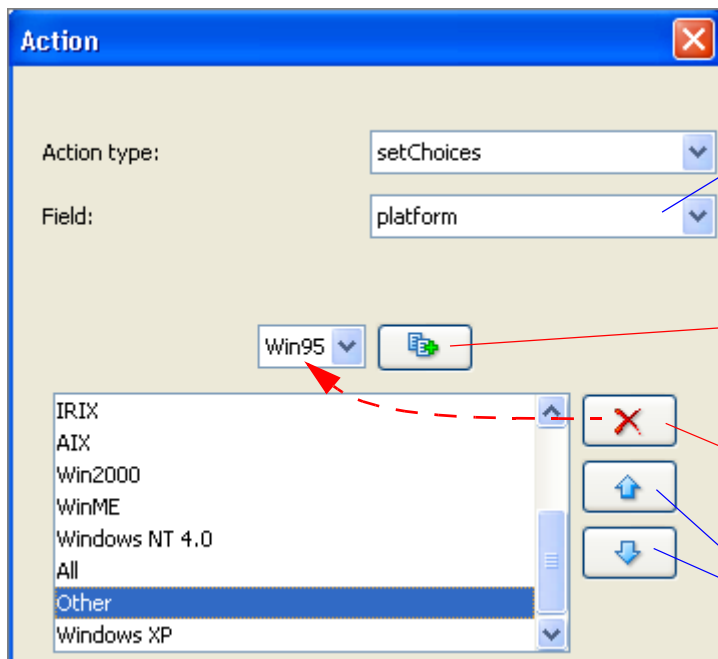
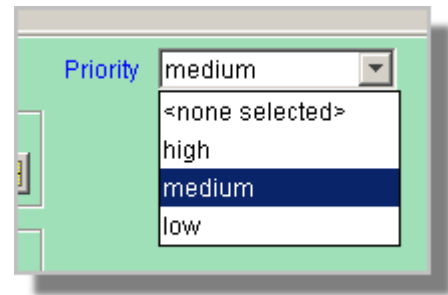
In a validation, a **setChoices** action can change the “choice list” that the user will see when he opens the list-box: The changes to the choice list can include:

- Removing one or more of the existing choices
- Changing the order of the choices

You cannot add new strings to the choice list with a **setChoices** action. The changes made by a **setChoices** action are temporary: they last only until the next **setChoices** action on the same field, or until the user closes the edit form. When an edit form is opened on another issue record (or subsequently on this issue record), the user will see the field’s original choice list, as defined on the **Schema** tab.

To define a **setChoices** action for a validation condition:

1. In the Action window, select **setChoices** as the action type, and use the **Field** list-box to select one of the database’s fields of type “choose”.
2. Use the controls at the bottom of the **Action** window to define the temporary change to the choice list:



select one of the fields of type “choose”

restore selected field from “temporarily deleted” list to choice list

remove selected field from choice list, sending it to “temporarily deleted” list

change order of choices

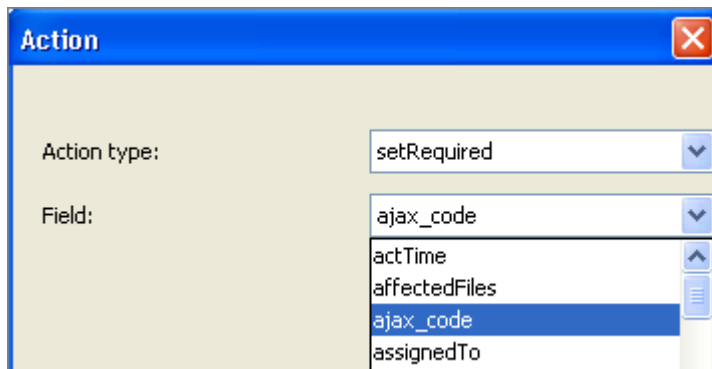
## Requiring a Value to be Entered in a Field

Like many other programs that process fill-in-the-blanks forms, AccuWork can treat certain fields as required fields — the user must specify a value in each such field before AccuWork will create or update an issue record. (For a “choose”, “list”, or “user” field, the value must not be the **<none selected>** placeholder.)

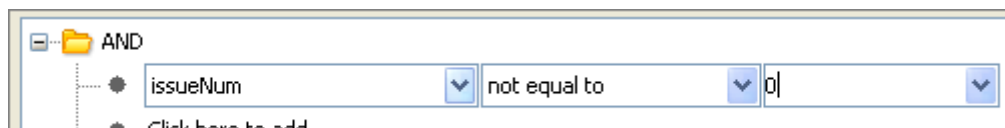
AccuWork affords you great flexibility in controlling required fields. A field’s “required” status is not part of the basic database schema. Instead, it is controlled by the conditional-validation facility. Thus, AccuWork repeatedly redecides whether a field is required: when the edit form first appears *and* each time the user changes any field value on the edit form. Whenever the user clicks the **Save** button, AccuWork uses the current set of required fields to allow or disallow the “save” operation.

To define a **setRequired** action for a validation condition:

1. In the Action window, select **setRequired** as the action type.
2. Select the field to be required.



If you want a field to be required in all circumstances, use the **setRequired** action along with a condition that’s always true. For example, you can exploit the fact that every issue record has a positive integer as its issue number:

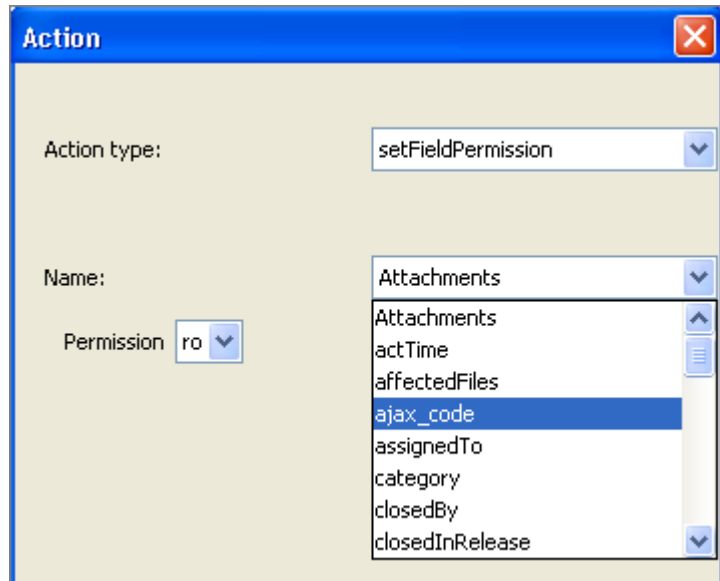


## Setting Permissions on All or Part of the Issue Record

The **setIssuePermission**, **setTabPermission**, and **setFieldPermission** actions are essentially similar: they restrict the editability of some component of the issue record:

- **setIssuePermission** makes the entire issue record read-only.
- **setTabPermission** makes a particular page (tab) of the issue record read-only. See *Designing an Edit Form* on page 194.
- **setFieldPermission** makes a particular field of the issue record read-only.

The “read-only” (**ro**) permission is the only one you can set with these commands. This setting affects the user’s current access to the issue record; the “read-only” status is not stored in the repository as part of any issue record. For example, user **allison** might find that she cannot change any fields on the Assignment page of any issue record, because of this validation:



AND CUR_USER is allison	setTabPermission( Assignment , "ro")
----------------------------	--------------------------------------

This restriction affect **allison**’s access to issue records; other users’ access remains unaffected. And if this conditional validation is subsequently removed, **allison** will regain access to the Assignment page of all the issue records.

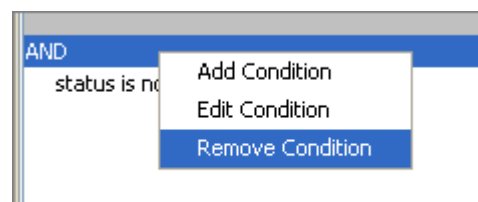
## Requiring Change Set Entries

The **setChangesRequired** action specifies that the user cannot **Save** an issue unless there is at least one entry in the issue record’s change set (Changes tab).

Note: be careful *not* to specify this action with the “initialization of new issue” condition. See *Initializing Field Values in a New Issue Record* on page 200.

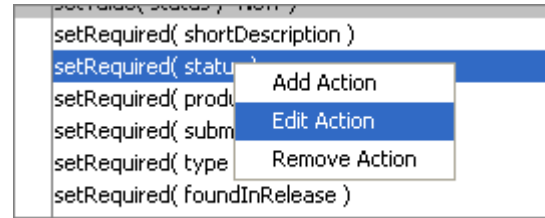
## Revising and Removing Validations and Actions

Each conditional validation consists of a condition (left column) and a set of actions (right column). To revise or remove an entire validation, right-click anywhere within the condition, and select the appropriate command, **Edit Condition** or **Remove Condition**, from the context menu.



For the first (unconditional) validation, “initialization of new issue”, you cannot remove or revise the condition. You can only work with the validation’s actions.

To revise or remove an individual action from a validation, right-click on that action and select the appropriate command, **Edit Action** or **Remove Action**, from the context menu.



## The Change Packages Tab

The AccuRev Enterprise version of AccuWork supports an enhanced integration between AccuRev and AccuWork: any AccuWork issue record can act as a change package, keeping track of which versions were created to fix the bug (or implement the new feature) described in that issue record.

In the Schema Editor, the Change Packages subtab provides controls for certain aspects of this facility. For details, see *Viewing Stream Contents/Differences in Terms of Change Packages* on page 217.



# Change Packages and Integrations between Configuration Management and Issue Management

Any version-control system must be able to keep track of the changes that developers make to individual files. A full-fledged configuration management system, like AccuRev, should be able to handle questions like these:

“What were all the changes made to source files in order to fix bug #457?”

“Have all the changes made to fix bug #457 been handed off to the QA Group” (That is, have the appropriate versions been promoted to the QA stream?)

AccuRev can handle such questions through its change package facility. (Change packages are available in the AccuRev Enterprise product only.)

## Structure of a Change Package

A change package is a collection of element versions; for example:

```
version kestrel_dvt_jjp/13 of element ./src/brass.c  
version kestrel_dvt_jjp/14 of element ./src/brass.h  
version kestrel_dvt_jjp/16 of element ./src/commands.c
```

The basic idea is that this set (or “package”) of versions contains all the changes required to implement a certain development project. But we need to refine this idea. Consider that version 14 of **brass.h** probably contains *more* than just the changes for that development project. For example:

- Versions 1-7 might have been created years ago, when the product was first developed
- Versions 8 and 9 might have been minor tweaks, performed last month
- Versions 10-14 are the only versions with changes for the development project in question

So we need a way to express the idea that only the “recent changes” to **brass.h**, those in versions 10-14, are to be included in the change package. AccuRev accomplishes this by defining each change package entry using two versions: a user-specified head version and an older, automatically-determined basis version. The “recent changes” to be included in the change package were made by starting with the basis version (version 9 in this example) and **Keep**’ing one or more new versions (versions 10, 11, 12, 13, and 14 in this example).

In the AccuRev GUI, the head version of a change package entry is usually identified simply as the “Version”.

Note: the **Patch** command uses the same “recent changes” analysis to determine which changes in the “from” version are to be incorporated into the “to” version.

Where should the change package entry for **brass.h** be recorded? AccuRev already provides a mechanism for keeping track of development projects: the AccuWork issue-management facility.



Each project — fixing a bug, creating a new feature, etc. — is tracked by a particular AccuWork issue record. So it makes sense to implement change packages using issue records.

Each issue record includes a **Changes** section that acts as an “accumulator” for versions’ changes. Here’s how the above example of a change package would appear in an issue record’s edit form:

The screenshot shows the 'Changes' tab of an issue record edit form. The form has a blue header with fields for 'Short Description' (Default colors are bogus), 'Issue' (3), 'Status' (Scheduled), 'State' (Open), 'Severity' (D), and 'Priority' (1). Below the header is a tabbed interface with tabs for 'Basics', 'Assignment', 'Misc', 'Attachments', 'Resolution', 'Changes' (selected and circled in red), and 'Issue History'. Below the tabs is a table with the following data:

Name	In Folder	Version	Basis Version
brass.c	\src\	5/13	5/10
chap03.doc	\doc\	5/11	4/7
tools.readme	\tools\	5/9	12/3

This change package has entries for three elements:

- **brass.c:** The basis version, 5/10 was created in the user’s own workspace. This indicates that the user promoted 5/10 to the backing stream. AccuRev assumes that this change was for another task, not the one covered by this issue record. Then, the user turned his attention to the current task, creating additional versions up to and including 5/13.
- **chap03.doc:** This change began when the user updated his workspace, bringing in version 4/7 of the element (which had originally been created in another workspace, then was promoted to the backing stream). Then, the user created one or more versions in his own workspace, up to and including version 5/11.
- **tools.readme:** Similarly, this change began when the user updated his workspace, bringing in version 12/3, originally created in another workspace. The user created one or more versions in his workspace, ending with version 5/9.

Each change package can include at most one entry for a given element. This rule helps to ensure that the changes in a given change package are consistent with each other. See [Updating Change Package Entries](#) on page 214.

## Creating Change Package Entries

You can add entries to a change package manually: right-click a version in the File Browser, Version Browser, or History Browser, and then select the **Send to Issue** command from the context menu. The selected version becomes the head version of the change package entry; AccuRev automatically determines the corresponding basis version. As the examples above suggest, AccuRev uses an algorithm that determines the set of “recent changes” to the element, made in a single workspace.

In the Version Browser, a variant command, **Send to Issue (specifying basis)**, enables you to pick the basis version, rather than allowing AccuRev to determine it automatically.

You can also invoke the **Send to Issue** command on the Changes tab of an issue record. This copies an existing change package entry to a different change package (issue record).



AccuRev can record change package entries automatically, whenever the **Promote** command is invoked in a workspace. For example, suppose issue record #3 represents a particular bug (and its fix). Whenever a developer promotes one or more versions whose changes address that bug, he specifies issue #3 at a prompt. AccuRev automatically creates a change package entry in issue #3 for each promoted version.

Automatic recording of change package entries is enabled through an integration between configuration management and issue management. This change-package-level integration is separate from the transaction-level integration that has long been an AccuRev feature. For more on both these integrations, see [Integrations Between Configuration Management and Issue Management](#) on page 220.

## Complex Change Package Entries

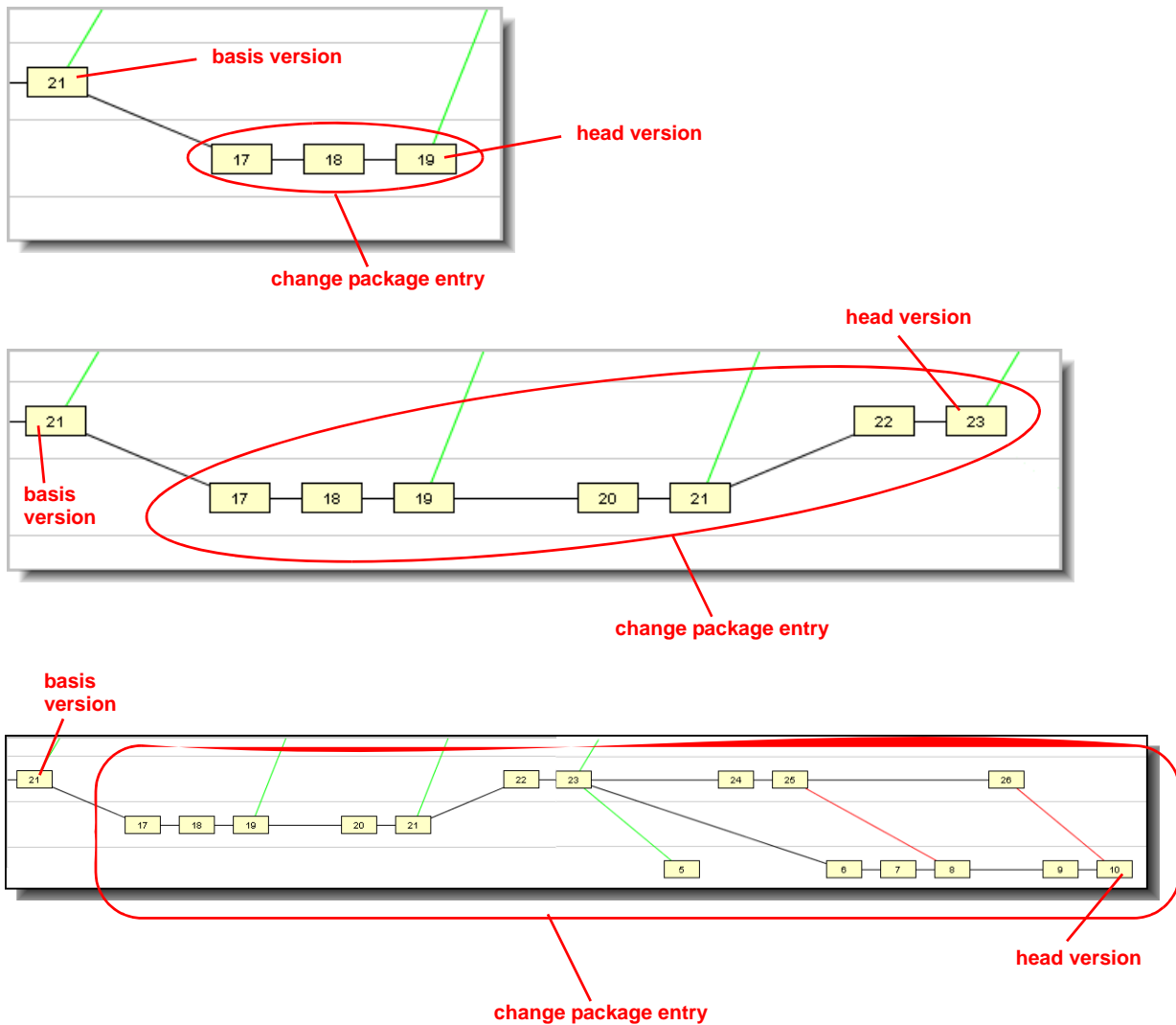
All change package entries are recorded in terms of real versions (those created in users’ workspaces), even though there may be corresponding virtual versions (created by promoting the real versions from workspaces to dynamic streams). In all the examples shown above, each change package entry is a series of consecutive real versions created in the same workspace — that is, each change package entry records a particular patch to the element.

But the change package facility can also track *ongoing* changes to elements — changes made at different times, and in different workspaces. To support this capability, AccuRev defines a change package entry in a more general way than a patch:

A change package entry for an element consists of all the real versions in the element’s version graph between a specified basis version and a specified head version. Between-ness is determined both by direct predecessor-successor connections (created, for example, by **Keep**)

and by merge connections (created by **Merge**). Patch connections are *not* considered in this determination; the basis version itself is not part of the change package entry.

The following Version Browser excerpts show the range of complexity that a change package entry can have. In fact, these excerpts show how the same change package entry can change over time, becoming more complex. See [Updating Change Package Entries](#) on page 214.

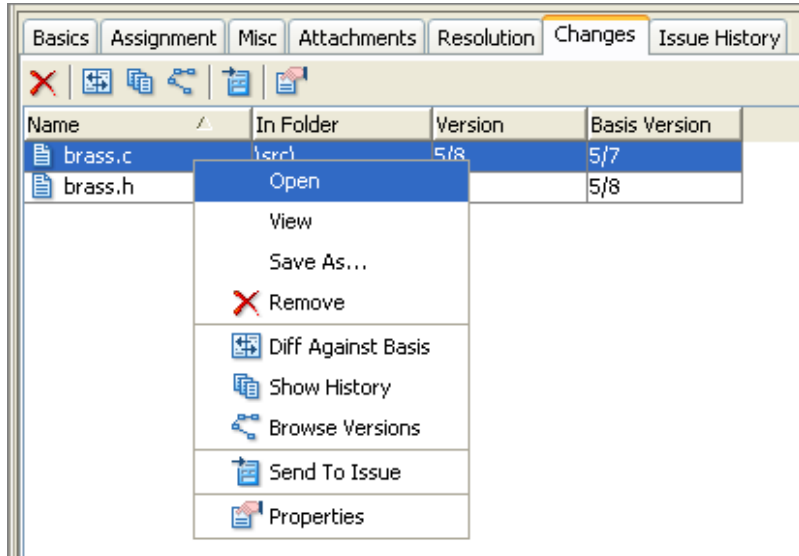


These illustrations suggest the following definition for a change package entry, which is equivalent to the definition above:

A change package entry for an element consists of the element's entire version graph up to the specified head version, *minus* the entire version graph up to the specified basis version. For these purposes, the version graph includes direct predecessor-successor connections and merge connections, but not patch connections.

## Operations on Change Package Entries

You can perform several operations on a change package entry, using its context menu. Most of these operations concentrate on the head version, which contains all the changes in the change package entry. (The head version appears in the “Version” column.)



### Open

Run the appropriate command on the head version of the change package entry, according to its file type.

### View

Open a text editor on a temporary copy of the head version (text files only).

### Save As

Copy the head version to another filename.

### Remove

Delete the selected entry(s) from the change package. This doesn't affect an entry's membership in other change packages, or the status of the element in the depot's stream hierarchy.

### Diff Against Basis

Use the graphical Diff tool to compare the head version with the basis version.

### Show History

Open a History Browser on the selected element.

### Browse Versions

Open a Version Browser on the selected element.

## Send to Issue

Copy the selected entry(s) to another issue record, which you specify in a pop-up window. With this command, you can make the same entry(s) can appear in two or more change packages.

The pop-up window offers the set of issue records selected by the default query of the issues database, but you can enter the number of any issue record. If the other issue record already has an entry for the element, AccuRev attempts to combine them. See *Updating Change Package Entries* on page 214.

## Properties

Displays information about the selected element: pathname, file type of current version, and element-ID.

The **Remove** and **Send To > Issue** commands enable you to maintain and fine-tune the contents of change packages. select the entry(s), invoke the **Send To > Issue** command, and specify another issue record. To remove one or more entries from a particular change package, just select them and invoke the **Remove** command.

## Updating Change Package Entries

Change packages are designed to track ongoing changes to elements, not just a single set of changes. This means there will be times when you want to add a change package entry for a particular element, but an entry for that element already exists in the change package. In such situations, AccuRev attempts to combine the new entry with the existing one, producing an updated entry that includes all the changes. (Recall that there can be at most one entry for a given element in a given change package.)

## A Little Bit of Notation

To help explain how AccuRev performs “change package arithmetic” to combine and update entries, we’ll use a simple notation. Suppose a change package entry contains the set of an element’s versions defined by these specifications:

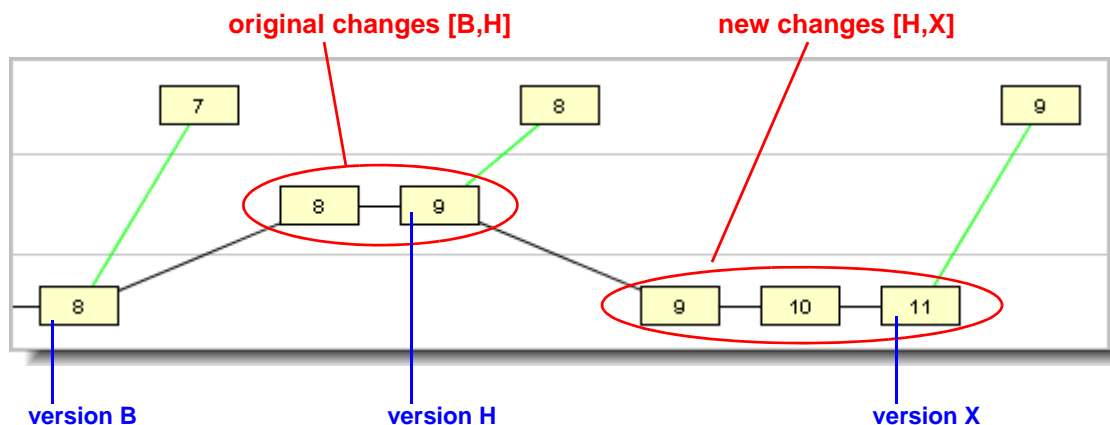
the head version is **H**  
the basis version is **B**

We’ll use the ordered pair **[B,H]** to indicate this change package entry.

## Combining Two Change Package Entries

Now, suppose a new change is to be combined with the existing change package entry **[B,H]**. There are several cases, each handled differently by AccuRev:

- **Case 1:  $[B,H] + [H,X]$**  — This simple case typically arises when you think you’re done with a task and record your work as change package entry **[B,H]** — but it turns out that more work on the same element is required. So you (or a colleague) start where you left off, with version **H**, and make changes up to version **X**. Then, you want to incorporate the new set of changes **[H,X]** into the same change package.

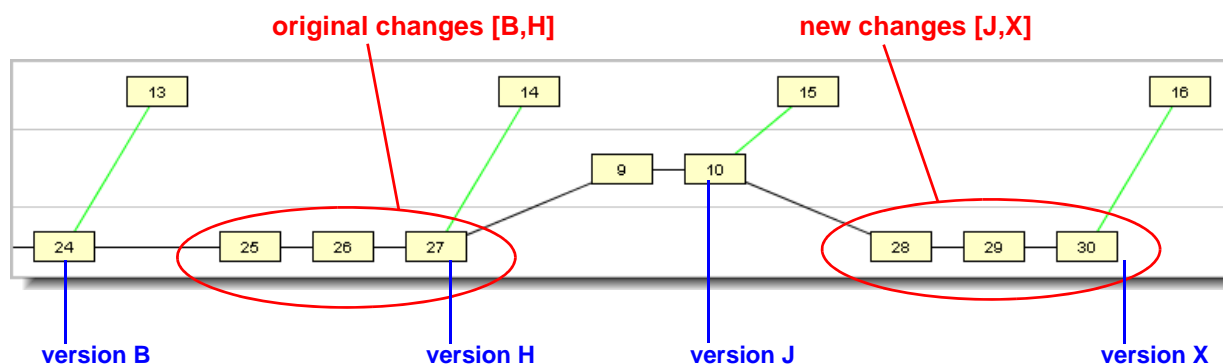


In this case, it's clear that the two series of changes can be viewed a single, uninterrupted series — starting at version B and ending with version X. That is:

$$[B,H] + [H,X] = [B,X]$$

Accordingly, AccuRev updates the change package entry automatically — keeping **B** as the “Basis Version” and changing the “Version” from **H** to **X**.

- **Case 2: [B,H] + [J,X]** (where H is an ancestor of J: “change package gap”) — This case typically arises when you do work on a task at two different times, and someone else has worked on the same element in between.

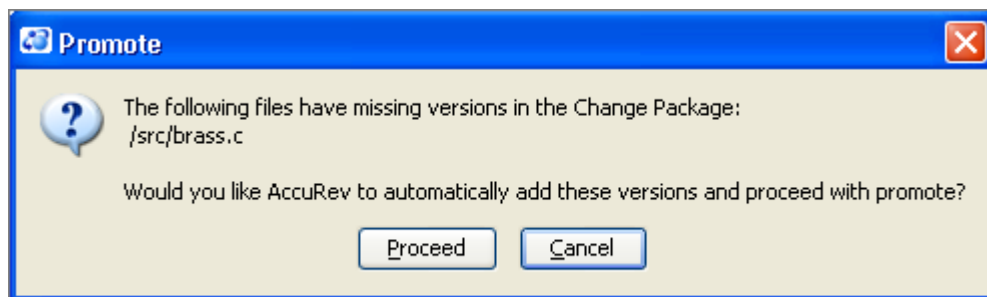


In this example, a colleague updated her workspace to bring in your original changes, created versions 9 and 10 in her workspace, and promoted her changes. You then updated your workspace to bring in her changes, and made a new set of changes.

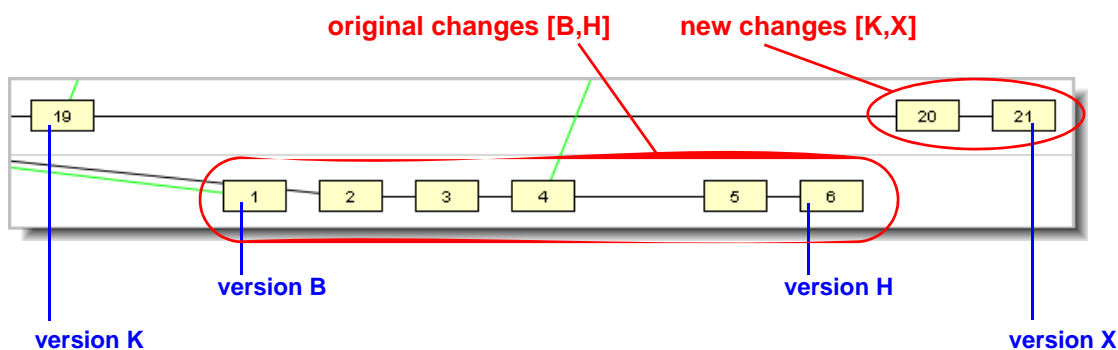
When AccuRev tries to combine the change **[B,H]** and the change **[J,X]** into a single change package entry, it detects that version **H** and version **J** are not the same, but that **H** is a direct ancestor of **J**. Thus, there is a simple “gap” in the potential combined change package entry (in this example, consisting of your colleague’s versions 9 and 10).

Probably, your colleague was not working on the same task when she made her changes. (If she had been, she would have added her changes to the same change package, as in Case 1.) On the other hand, it's probably OK to include the entire, uninterrupted series of versions **[B,X]** in your change set — this includes both your original changes and your new changes (and, harmlessly, your colleague’s changes, too).

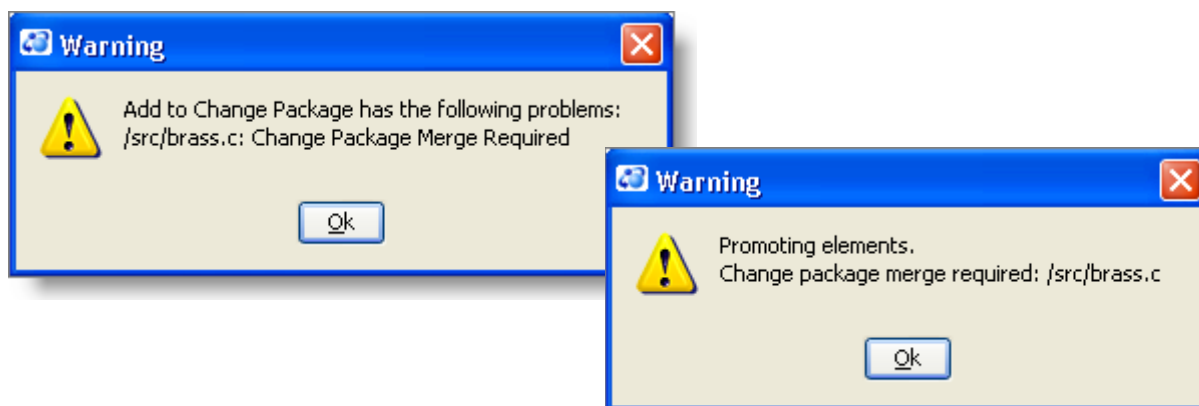
Accordingly, AccuRev prompts you to approve this “spanning the gap” between the two change set entries, in order to create a single, combined entry:



- **Case 3: [B,H] + [K,X]** (where H is *not* an ancestor of K: “change package merge required”) — This case typically arises when developers in workspaces that do not share the same backing stream try to use the same change package. There is no simple “gap” between the existing change package entry and the new one — which means there is no way to combine them into a single change package entry, according to definitions in *Complex Change Package Entries* on page 211.



AccuRev signals this situation with a “change package merge required” message, and cancels the current operation.



You can remedy this situation by performing a merge at the element level. (There is no merge operation defined at the change package level.) In the example above, merging version H and

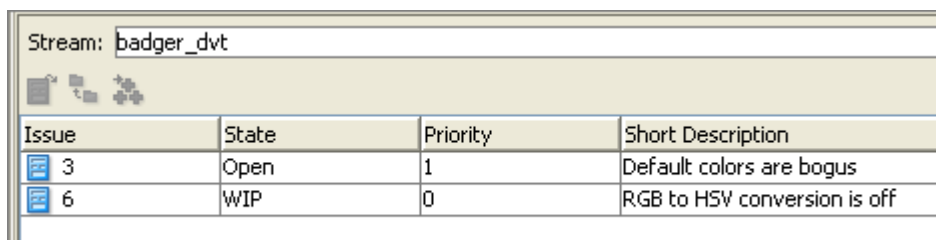
version X would create a new version; a change package entry with the new version as its head can be combined with the existing entry.

## Viewing Stream Contents/Differences in Terms of Change Packages

You have always been able to view the contents of a stream or workspace in terms of individual elements and versions (File Browser details pane). Likewise, you've been able to compare two workspaces/streams in terms of individual elements and versions (StreamBrowser **Diff** command).

With the advent of change packages, you can now view the contents of a single workspace/stream — or compare two workspaces/streams — in terms of change packages. The commands, available in the StreamBrowser, are **Show Issues** and **Show Difference > By Issues**. For both these commands, the result is a set of issue records. AccuRev displays the results similarly to the way it displays the results of a AccuWork query: a table in which each row is one issue record and each column is one field:

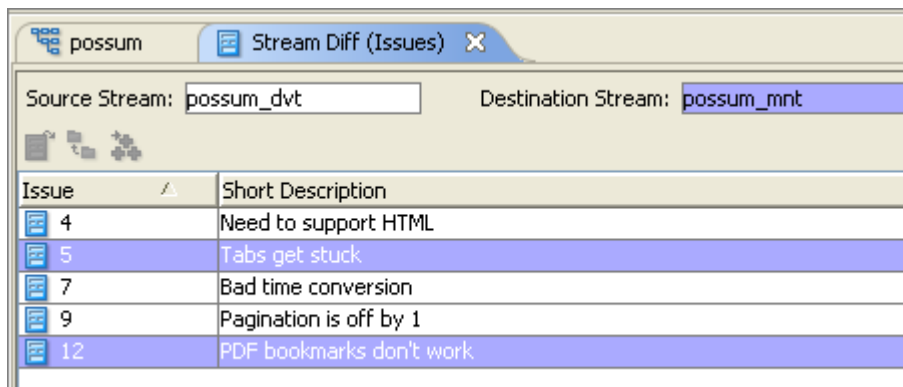
result of  
**Show Issues**



The screenshot shows a window titled 'Stream: badger\_dvt'. It contains a table with four columns: 'Issue', 'State', 'Priority', and 'Short Description'. There are two rows of data.

Issue	State	Priority	Short Description
3	Open	1	Default colors are bogus
6	WIP	0	RGB to HSV conversion is off

result of  
**Show Diff  
By Issues**



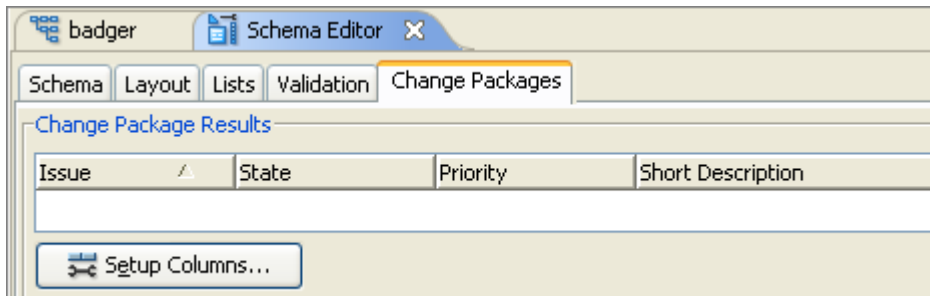
The screenshot shows a window titled 'possum' with a tab 'Stream Diff (Issues)'. It contains a table with two columns: 'Issue' and 'Short Description'. There are five rows of data.

Issue	Short Description
4	Need to support HTML
5	Tabs get stuck
7	Bad time conversion
9	Pagination is off by 1
12	PDF bookmarks don't work

## Formatting a Change Package Table

To specify the format of the tables produced by **Show Issues** and **Show Differences by Issues** commands, go to the Schema Editor's Change Packages subtab. In the top section, "Change Package Results", you can determine which fields appear as columns, the column widths, the order of the columns (fields), and the sort order of the rows (records).

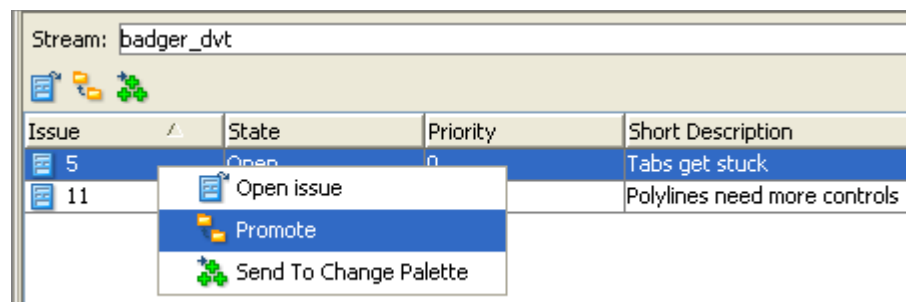




For more on these commands, see the *AccuRev User's Guide (GUI): Streams and Change Packages* on page 99, and *Comparing Streams Using Change Packages* on page 108.

## Promote by Change Package

You can invoke the **Promote** command on one or more of the issue records in a **Show Issues** table. This promotes all the versions in the issue record's change package to the parent stream.

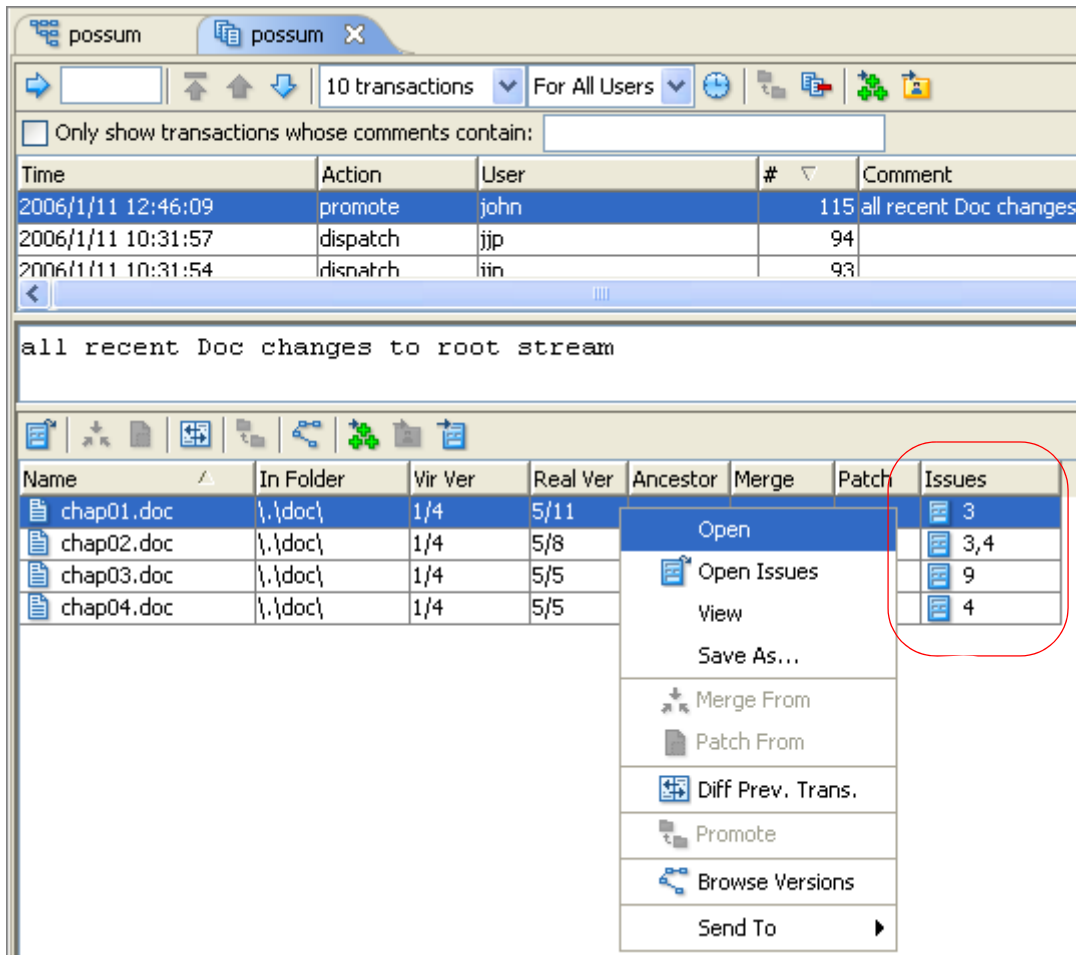


Promotion makes sense only for versions that are currently active in the stream. Accordingly, this command is restricted to issue records that are listed when **Show Active** is checked and **Show Incomplete** is not checked. A version in a change package won't be promoted if AccuRev determines that the changes in that version have already been propagated to the parent stream.

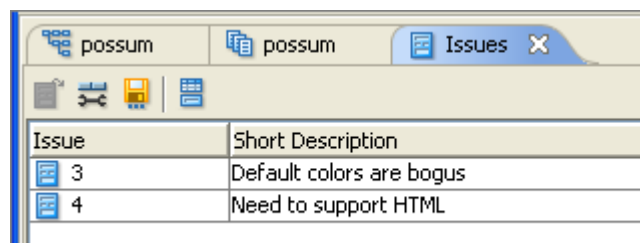
The **Promote** command is not enabled for issue records displayed by **Show Difference By Issues**.

## Viewing a Transaction in Terms of Change Packages

The Versions pane of the History Browser shows information on all the versions involved in a selected transaction. This pane includes an Issues column, which indicates the change package(s) to which each version belongs.



The **Open Issues** command, on the context menu of a version, enables you to view the entire change package(s) — that is, the issue record(s). If you select a version that belongs to a single change package, an edit form opens on that one issue record. If you select a version that belongs to multiple change packages, an Issues tab opens, listing all the issue records.



## “Locking” a Change Package

You can use AccuWork’s edit form validation facility to implement locking of change packages. This scheme relies of these facts:

- Each change package is the Changes tab of a particular issue record.
- Any tab of an issue record can be set to read-only status with the **setTabPermission** action in a validation.

For example, a validation could specify that the Changes tab's permission be set to **ro** when an issue record's Status changes to **Closed**. See *Defining Edit Form Validations* on page 198 and *Setting Permissions on All or Part of the Issue Record* on page 205.

## Integrations Between Configuration Management and Issue Management

This section describes two similar, but separate facilities that integrate AccuRev's configuration management functionality with its issue management (AccuWork) functionality. One of them uses change packages as the point of integration. The other uses a particular issue-record field as the point of integration. Both integrations record information about the **Promote** transaction in a user-specified AccuWork issue record.

### Change-Package-Level Integration

When a **promote** command is executed a user's workspace (but not in a higher-level dynamic stream), the change-package-level integration records all the promoted versions on the Changes subtab of a user-specified AccuWork issue record.

#### Enabling the Integration

The change-package-level integration is enabled on a depot-by-depot basis. Open the AccuWork Schema Editor for a particular depot's issue database, and go to the Change Packages subtab. Filling in the lower section, "Change Package Triggers", enables the integration for that particular depot.

The Change Package Triggers section is structured as a set of condition/query pairs. One of the queries will be selected for execution at **promote**-time. If the first condition is satisfied, the first query will be executed; otherwise the second condition will be evaluated, and if it's satisfied, the second query will be executed; and so on.

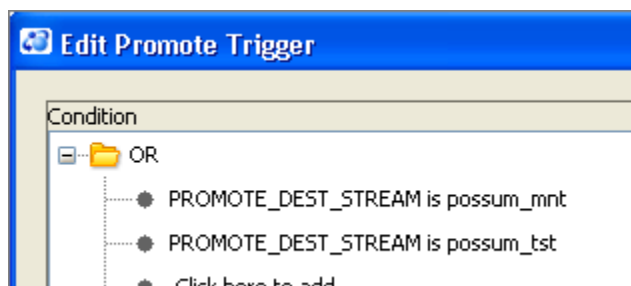
The screenshot shows the 'Schema Editor' window with the 'Change Packages' tab selected. It contains two main sections: 'Change Package Results' and 'Change Package Triggers'.

**Change Package Results:** This section has a table with columns 'Issue' and 'Short Description'. Below the table is a 'Setup Columns...' button. A red bracket on the right points to this section with the text: "results format for **Show Issues** and **Show Diff By Issues** commands".

**Change Package Triggers:** This section contains a table with two columns: 'Conditions' and 'Queries'. It lists two conditions, each with an associated query. A red bracket on the right points to this section with the text: "one of these queries will be executed at **Promote-time**".

Below the triggers table is another table with columns 'Issue', 'Status', 'Priority', and 'Short Description', and a 'Setup Columns...' button. A red bracket on the right points to this section with the text: "and its results will be displayed in this format".

Each clause of a condition performs a test on the **Promote** destination stream. For example, this condition is satisfied if the user is promoting to either of the streams **possum\_mnt** or **possum\_tst**:



The query corresponding to each condition can be any AccuWork query, which selects a set of issue records. See [Using AccuWork Queries](#) on page 174.

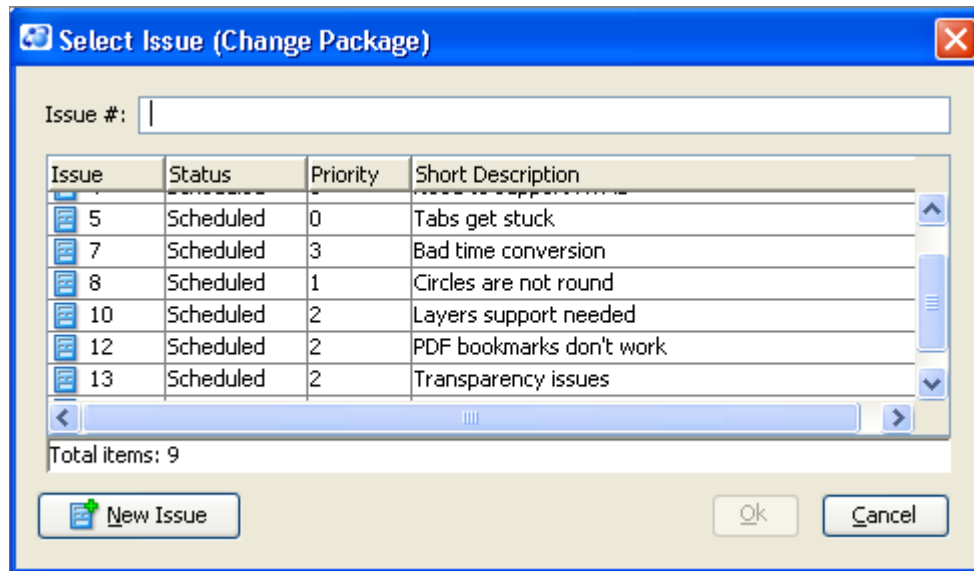
The Change Package Triggers section also specifies the format of each query's results table — the fields to appear as columns, the column widths, the order of the columns (fields), and the sort order of the rows (records).

## Triggering the Integration

Once the integration is enabled for a depot, it is triggered whenever a user performs a **promote** command in a workspace associated with that depot.

If the **Promote** command is invoked through the AccuRev GUI:

1. One of the AccuWork queries specified in the Change Package Triggers section is executed, selecting a certain set of issue records.
2. Those records are displayed to the user in the results table format specified in the Change Package Triggers section.
3. The user selects one or more of the issue records. There is also a **New Issue** button, which enables the user to create a new issue record “on the fly”.



4. The command completes its work.
5. The versions involved in the command are recorded in the change package section (Changes page) of the selected issue record(s).

If the **promote** command is invoked through the AccuRev CLI:

1. Just as with the GUI, one of the AccuWork queries specified in the Change Package Triggers section is executed, selecting a certain set of issue records.
2. The user is prompted to specify an issue record:

Please enter issue number ?

Users can bypass this prompt by specifying an issue number with the **-I** option:

```
> accurev promote -I 4761 chap01.doc
Validating elements.
Promoting elements.
Promoted element \.\doc\chap01.doc
```

Note: attempting to update an existing change package entry might cause a “change package gap” or “change package merge required” situation, either of which cancel the **promote** command. For example:

```
Promoting elements.
Change package gap: /doc/chap01.doc
```

You can handle a change package gap by adding the **-g** option to the **promote** command. This combines the new and existing change package entries by “spanning the gap”:

```
> accurev promote -I 4761 -g chap01.doc
Validating elements.
Promoting elements.
Promoted element \.\doc\chap01.doc
```

There is no way to have the **promote** command automatically handle a “merge required” situation. You must either perform a merge on the element to be promoted, or remove the existing change package entry for that element.

For more on “change package gap” and “change package merge required” situations, see [Updating Change Package Entries](#) on page 214 above.

3. Assuming the user-specified issue record is one of those selected by the query, the command completes its work, and the promoted versions are recorded in the change package section of the selected issue record.

What happens if the user specifies no issue record or a non-existent issue record, such as 99999 or 0? In both the GUI and CLI cases:

- If the user enters “0” (or equivalently, makes a blank or non-numeric entry), AccuRev checks whether issue record #0 is among the issues selected by the query executed in Step 1.

Note: the query *can* select issue record #0, even though it doesn’t exist — for example with this clause:

```
issueNum equal to 0
```

- If the query does select issue record #0, the user’s command completes but no information is sent to the issues database. This provides a way for the user to bypass the integration when performing the **promote** command.
- If the query does not select issue record #0, the user’s command is cancelled, and no information is sent to the issues database.
- If the user specifies a non-existent issue record, such as “99999”, the command is cancelled, and no information is sent to the issues database.

## Transaction-Level Integration

The integration between configuration management and issue management at the transaction level records the number of each **promote** transaction in a particular field of a user-specified issue record.

### Enabling the Integration

The transaction-level integration is enabled on a depot-by-depot basis, by setting the depot’s **pre-promote-trig** trigger. For example:

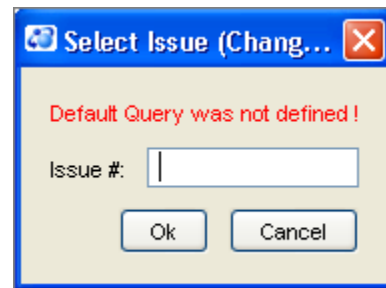
```
accurev mktrig -p kestrel pre-promote-trig client_dispatch_promote
```

Note that “client\_dispatch\_promote” is simply a keyword, not the name of a script file. The integration is implemented by two cooperating routines, one built into the AccuRev client software, one built into the AccuRev server software.

## Triggering the Integration

Once the integration is enabled for a depot, it is activated whenever a user executes the **Promote** command in any workspace or dynamic stream.

1. The depot’s default query, as defined on the Queries tab (**Issues > Queries**), is executed and the results are displayed to the user.
2. The user selects one of the issue records. Note that if no default query is defined for the depot, the user is prompted to type an issue record number.
3. The **promote** command completes its work, propagating the versions to the backing stream.
4. The **promote** transaction number is recorded in the **affectedFiles** field of the selected issue record. (This change to the issue record is, itself, recorded as a transaction, of kind **dispatch**.)



If the user enters “0” or makes a blank entry, the **promote** command completes but no change is made to any issue record. This provides a way for the user to bypass the integration.

Over time, the **affectedFiles** field of a given issue record can accumulate a SPACE-separated list of **Promote** transaction numbers.

## Implementation and Customization of the Transaction-Level Integration

When the transaction-level integration is activated, processing takes place on both the AccuRev client machine and the AccuRev server machine:

- The client-side processing — querying the AccuWork issues database and prompting the user to specify an issue record — is structured as a **pre-promote-trig** trigger routine built into the AccuRev client software.
- The server-side processing — updating of the AccuWork issue record — is structured as a **server-post-promote-trig** trigger routine built into the AccuRev server software.

You enable the integration by setting the **pre-promote-trig** trigger with the “client\_dispatch\_promote” keyword, as described above. You don’t need to explicitly set a **server-post-promote-trig** trigger script. If you do, the script runs instead of — not in addition to — the server-side built-in routine.

In most cases, you’ll want to avoid setting a **server-post-promote-trig** trigger script, just letting the built-in routines do their work. But suppose that after a **Promote**, you want the server machine to perform operations in addition to those defined in the transaction-level integration — for example, updating reference trees and sending email messages. In such cases:

1. Create a script that performs the server-side part of the transaction-level integration, along with the desired additional processing. Start with the sample script **server\_dispatch\_promote\_custom.pl**, which is located in the **examples/dispatch** subdirectory of the AccuRev installation directory.
2. Place the script in the AccuRev **bin** directory.
3. Use a **mktrig** command to make the script the depot's **server-post-promote-trig** trigger script.

Further customizations of the transaction-level integration are possible. For example, you might want the user to be able to specify several issue records, not just one. Or you might want to link **promote** commands in one depot with the AccuWork issues database in another depot. Or you might want to update an issue record field other than **affectedFiles**. In such cases, you'll want to dispense with the built-in "client\_dispatch\_promote" routines altogether:

1. Start with the sample script **client\_dispatch\_promote\_custom.pl** (in the **examples/dispatch** subdirectory), and create your own script for use as a **pre-promote-trig** script to execute on the client.
2. As described above, start with the sample script **server\_dispatch\_promote\_custom.pl** (in the **examples/dispatch** subdirectory), and create your own script for use as a **server-post-promote-trig** script to execute on the server.

## If Both Integrations are Enabled

Both the change-package-level and transaction-level integrations can be enabled for a given depot at the same time. In this case, a user performing a **Promote** command in a workspace is prompted to specify an issue record just once, not twice. The prompting for an issue record by the change-package-level integration takes place as usual. That issue record is then updated by both integrations.

Note that even if both integrations are enabled, a **Promote** command performed in a dynamic stream (not a workspace) activates just the transaction-level integration, not the change-package-level integration.





# AccuWork Command-Line Interface

This note describes aspects of the command-line interface to the AccuWork issue management system.

Note: the information herein is accurate as of Version 4.0, but this interface might change or be discontinued in a future release.

## Overview

The AccuWork CLI is implemented through a single command, **accurev xml**. The **xml** command is a non-interactive general-purpose command dispatcher; it reads a specified file to determine the AccuRev operation — in this case, a AccuWork command — to be invoked. For example:

```
accurev xml -l mycmd.xml           (“dash-ell” not “dash-one”)
```

Here, the **xml** command’s input comes from a file, **mycmd.xml**, which must contain an XML document. (The filename is irrelevant, and need not have a **.xml** suffix.) The XML document might contain this specification of a AccuWork query:

```
<queryIssue
  issueDB="UserReportedBugs"
  useAlt="false"
  expandUsers="true">
21 == "rel2.0"
</queryIssue>
```

This example specifies the command, “find all issue records in depot **UserReportedBugs** whose value in field #21 (the **targetRelease** field) is the string **rel2.0**”. The results of an **xml** command are sent to standard output, also in the form of an XML document. For example, this query might retrieve two issue records, producing this output:

```
<issues>
  <issue>
    <issueNum
      fid="1">2</issueNum>
    <transNum
      fid="2">3</transNum>
    <targetRelease
      fid="21">rel2.0</targetRelease>
    <type
      fid="7">defect</type>

    ... additional fields ...

    <platform
      fid="12">All</platform>
  </issue>
```

```

<issue>
  <issueNum
    fid="1">3</issueNum>
  <transNum
    fid="2">26</transNum>
  <targetRelease
    fid="21">rel2.0</targetRelease>
  <type
    fid="7">enhancement</type>

... additional fields ...

  <platform
    fid="12">Linux</platform>
</issue>
</issues>

```

This output provides the correspondence between field-ID numbers (e.g. fid="21") and field-names (e.g. targetRelease). This correspondence is important, since you must specify a query using field-IDs, not field-names (e.g. 21 == "rel2.0", not targetRelease == "rel2.0").

## AccuWork CLI Operations

You can perform the following AccuWork operations through the command-line interface:

- Query an issues database (<queryIssue> document) — Retrieve the contents of all issue records in a particular depot (issues database) that match a specified query.
- Create a new issue record (<newIssue> document) — Enter a single new issue record in a particular depot.
- Modify an existing issue record (<modifyIssue> document) — Change the contents of a single issue record that already exists in a particular depot.

The sections below provide guidelines for performing each of these operations. But there's an important prerequisite step to perform first

## Determining the Field-ID / Field-Name Correspondence

In the AccuWork CLI, you identify a field by its field-ID, not by its field-name. Thus, before doing any real AccuWork CLI work, you must determine the correspondence between field-IDs and field-names in your depot (issues database). This information is stored in the schema configuration file on the AccuRev Server host:

```
<AccuRev-inst-dir>/storage/depots/<depot-name>/dispatch/config/schema.xml
```

You can retrieve the contents of the **schema.xml** file from an AccuRev client machine with this command:

```
accurev getconfig -p <depot-name> -r schema.xml
```

Extract the **name=** and **fid=** text lines from this data, and store them for future reference. You'll need to refer to this information often as you work with the AccuWork CLI. Let's call this extracted data the field-ID definitions.

For example, the field-ID definitions for the default schema look like this:

```
name="issueNum"
fid="1">
name="transNum"
fid="2">
name="status"
fid="3">
name="shortDescription"
fid="4">
name="state"
fid="5">
name="productType"
fid="6">
name="type"
fid="7">
name="severity"
fid="8">
name="priority"
fid="9">
name="submittedBy"
fid="10">
name="dateSubmitted"
fid="11">
name="platform"
fid="12">
...
```

This data shows that field **submittedBy** has field-ID **10**, field **productType** has field-ID **6**, etc.

Note: all examples in the remainder of this document will use the field-ID/field-name correspondence in the above example.

The contents of these XML elements are the field values for issue record #1. A couple of them, **submittedBy** and **dateSubmitted**, have values that might be a bit surprising — numbers instead of strings. We'll discuss these kinds of values in section *Selecting Issue Records with a Query* below.

## Selecting Issue Records with a Query

The simplest kind of query selects one or more issue records by comparing the value of one field with a literal value (a constant) — for example, “is the value of the issueNum field equal to 415?” or “does the shortDescription field contain the string ‘busted’?”.

For such simple queries, you can adapt the one we used in section *Determining the Field-ID / Field-Name Correspondence* above. This query finds all issue records whose **productType** value is **Frammis**.

```
<queryIssue issueDB="XXXXX" useAlt="false">
  6 == "Frammis"
</queryIssue>
```

The output of a query is an XML document whose top-level <issues> element contains zero or more <issue> sub-elements:

```
>>> accurev xml -l query-filename
<issues>
  <issue>
    ... individual field-name elements ...
  </issue>
  <issue>
    ... individual field-name elements ...
  </issue>
  ...
</issues>
```

## More Complex Queries

You can compose and run arbitrarily complex queries. If the query goes just a bit beyond the simplest, you can probably compose it manually. For example, this query finds all issue records whose **productType** value is **Frammis** or **Widget**:

```
<queryIssue issueDB = "dpt38" useAlt = "false" useAltQuery = "false">
  <OR>
    <condition>6 == "Frammis"</condition>
    <condition>6 == "Widget"</condition>
  </OR>
</queryIssue>
```

With more complex queries, be sure to include the useAltQuery = "false" attribute in the <queryIssue> start-tag.

But to minimize the chance of getting lost in the syntax of more complex queries, we strongly recommend that you compose the complex query graphically, then “export” the query to a text file:

1. In the AccuRev GUI, enter the command **Issues > Queries** to enter the Query Editor.

2. Create a new query, give it a name, and specify the query logic.
3. Click the **Save All Queries** button.
4. Place an XML-format dump of *all* your queries in a text file:

```
accurev getconfig -p depot-name -u user-name -r query.xml > myqueries.txt
```

This **getconfig** command retrieves the contents of the configuration file that stores your queries:

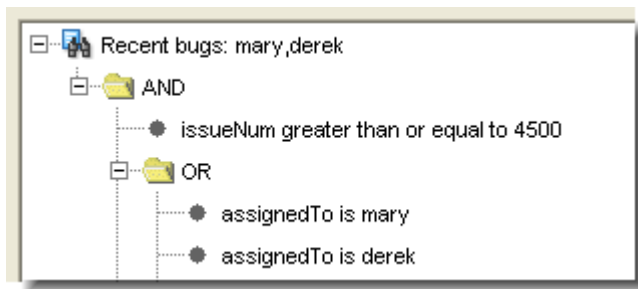
*AccuRev-install-dir/storage/depots/depot-name/dispatch/config/user/user-name/query.xml*

5. Use a text editor to extract the XML `<queryIssue>` element that defines the query. (Don't extract the entire `<query>` element, which includes the query's name and output field definitions.)
6. Store the `<queryIssue>` element code in a separate file, say **framemis\_or\_widget.xml**.

You can now invoke the query with the CLI:

```
accurev xml -l framemis_or_widget.xml
```

Example: This GUI-composed query ...



... is represented by this XML code:

```
<queryIssue issueDB = "dpt38" useAlt = "false" useAltQuery = "false">
<AND>

<condition>
1 >= &quot;4500&quot;;

</condition>
<OR>

<condition>
14 == &quot;2&quot;;

</condition>

<condition>
14 == &quot;24&quot;;
```

```

</condition>
</OR>
</AND>
</queryIssue>

```

This code is perfectly good XML, even though it's not "pretty-printed". Also note the use of the XML entity reference **&quot;**; which is equivalent to a double-quote character.

## Special Field Types

Each field in a AccuWork issues database has a field type: Text, Choose, List, etc. For a field of type User (such as the **submittedBy** field in the example in section *Determining the Field-ID / Field-Name Correspondence* above), a query defaults to reporting users by their integer user-IDs, rather than by their usernames (principal-names):

```

...
<submittedBy
  fid="10">40</submittedBy>
...

```

In this case, you could use the output of the **accurev show users** command to determine that user **jjp** has user-ID **40**. Alternatively, you can modify the query to set the attribute `expandUsers` in the `<queryIssue>` start-tag:

```

<queryIssue issueDB = "dpt38"
  useAlt = "false"
  useAltQuery = "false"
  expandUsers = "true">
...

```

Setting `expandUsers` causes the query to report the values of User fields with usernames instead of user-IDs:

```

...
<submittedBy
  fid="10">jjp</submittedBy>
...

```

For a field of type Timestamp (such as the **dateSubmitted** field in the example in section *Determining the Field-ID / Field-Name Correspondence* above), a query always reports the timestamp as a large integer, representing the number of seconds since Jan 1, 1970 UTC:

```

...
<dateSubmitted
  fid="11">1083606273</dateSubmitted>
...

```

(This is the standard Unix timestamp scheme.) You can use Perl to convert the integer into a human-readable string:

```

>>> perl -e "print scalar localtime(1083606273)"
Mon May  3 13:44:33 2004

```

## Creating a New Issue Record

To create a new issue record in a particular issues database, execute the command

```
accurev xml -l my_datafile
```

... where **my\_datafile** contains an XML document in this format:

```
<newIssue issueDB="XXXXX">
  <issue>
    ... individual field-value specifications ...
  </issue>
</newIssue>
```

As always, replace **XXXXX** with the name of the depot that stores the issues database. For the individual field-value specifications, adapt the output of a query that retrieves a single issue record. The complete contents of **my\_datafile** might be:

```
<newIssue issueDB="UserReportedBugs">
  <issue>
    <type
      fid="7">defect</type>
    <submittedBy
      fid="10">5</submittedBy>
    <foundInRelease
      fid="20">rel2.0</foundInRelease>
    <productType
      fid="6">Widget</productType>
    <shortDescription
      fid="4">Names are sometimes trunca</shortDescription>
    <dateSubmitted
      fid="11">1083606275</dateSubmitted>  </issue>
  </newIssue>
```

Some DOs and DON'Ts:

- You must specify the value of a User field with a user-ID (**5**), not a username (**derek**).
- You must specify the value of a Timestamp field with a number-of-seconds integer, not a string. You can use the **timelocal()** function in the Perl module **Time::Local** to generate these integers:

```
use Time::Local;
$sec = 35;    # range = 0 .. 59
$min = 22;    # range = 0 .. 59
$hr  = 14;    # range = 0 .. 23
$dte = 8;     # range = 1 .. 31
$nth = 5;     # range = 0 .. 11 (January is the zero'th month!)
$yr  = 2004;  # play it safe: use a 4-digit number
$numseconds = timelocal($sec, $min, $hr, $dte, $nth, $yr);
print $numseconds, "\n";
```



- Don't include specifications for the **issueNum** and **transNum** fields. AccuWork assigns these values automatically.
- The field initialization and validation code defined in the issues database schema will not be executed when the issue record is created. It's up to you to specify the appropriate values for the appropriate fields.

The validations *will* be invoked when the issue record is subsequently opened in the AccuWork GUI. In particular, you can create an issue record with a List field whose value is not in the field's list of possible values. But when the AccuWork GUI opens the issue record, it replaces the bogus value with <none selected>.

- Be sure that the top-level and second-level XML elements are named <newIssue> and <issue>. The names for the individual-field elements are irrelevant — only the **fid** attributes count. The following specifications are equivalent:

```
<status fid="20">New</status>
<myfield fid="20">rel2.0</myfield>
```

When you submit the <newIssue> data structure to the AccuWork CLI, it creates the record and reports the new record's contents. This report includes the automatically assigned **issueNum** and **transNum** values:

```
>>> accurev xml -l my_datafile
<issue>
  <issueNum
    fid="1">11</issueNum>
  <transNum
    fid="2">154</transNum>
  <type
    fid="7">defect</type>
  <submittedBy
    fid="10">24</submittedBy>
  <foundInRelease
    fid="20">rel2.0</foundInRelease>
  <productType
    fid="6">Frammis</productType>
  <shortDescription
    fid="4">Refuses to fram</shortDescription>
  <dateSubmitted
    fid="11">1062787292</dateSubmitted>
</issue>
```

See also *Using 'modifyIssue' to Create New Issue Records* on page 235 below.

## Modifying an Existing Issue Record

Use the following procedure to modify an existing issue record:

1. Create a query to select the desired issue record, as described in *Selecting Issue Records with a Query* on page 230. For example, to select issue record #472 from the **Problems** issues database, create this XML document:

```
<queryIssue issueDB="Problems" useAlt="false">
  1 == "472"
</queryIssue>
```

2. Run the query, storing the results in a text file:

```
accurev xml -l myquery.xml > issue472.xml
```

3. Edit the text file, making these changes:

- Change the top-level XML **<issues>** start-tag to **<modifyIssue>**. Include **issueDB** and **useAlt** attributes in the start-tag:  

```
<modifyIssue issueDB = "Problems" useAlt="false">
```
- At the end of the file, change the **</issues>** end-tag to **</modifyIssue>**.
- Remove the entire **<transNum>** element.
- Change the values of one or more existing fields.
- If you wish, add new field values, using the discussion in *Creating a New Issue Record* on page 233 as a guide.

4. Submit the edited text file:

```
accurev xml -l issue472.xml
```

When you submit the **<modifyIssue>** data structure to the AccuWork CLI, it modifies the specified issue record. As when you create a new issue record with the AccuWork CLI, field validations are not applied.

To verify the new record's contents, submit the original query again:

```
accurev xml -l myquery.xml
```

## Using 'modifyIssue' to Create New Issue Records

Instead of using a **<modifyIssue>** document to change an existing issue record, you can use it to create a new issue record. This is useful for copying issue records from one issues database to another. For example, you might use a **<queryIssue>** document, as described earlier, to retrieve the contents of an issue record from database **Problems**, in the form of an **<issues>** document. As described in Step 3 above, when you change the **<issues>** tag to a **<modifyIssue>** tag, specify a different database:

```
<modifyIssue issueDB = "Problems_Public" useAlt="false">
```

In this example, the issue record will effectively be copied from the **Problems** issues database to the **Problems\_Public** issues database. Make sure the target database exist and has the same schema as the source database.

The **<modifyIssue>** technique is also useful for making copies of the issue records that act as change packages, and for replicating issue records between different AccuRev sites.

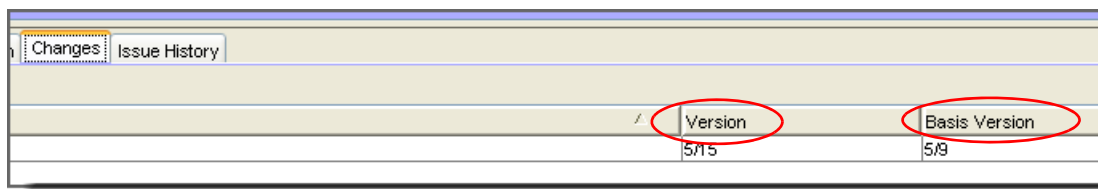
Note that a new issue record created with **<newIssue>** always gets assigned the next available issue number in the database; by contrast, a new issue record created with **<modifyIssue>** gets the issue number specified by the **<issueNum fid="1">** subelement:

```
<issueNum
  fid="1">4197</issueNum>
```

An issue record with this number must not already exist in the target database, but there is no other restriction. It's perfectly OK to have a “sparse” issues database, in which most issue numbers are unallocated.

## Interface to the Change Package Facility

AccuWork issue records are used to implement the change package facility. The set of changes in a change package is indicated by a set of “Versions”, listed on the Changes subtab of an issue record, each with a corresponding “Basis Version”:



Version	Basis Version
5/15	5/9

Each Version / Basis Version pair defines a set of changes to the element: the changes made since the Basis Version was created, up to and including the Version. The change package consists of such Version / Basis Version “change intervals” for any number of elements.

Various user commands and triggers maintain the contents of a change package: adding new versions and removing existing versions. There are also commands for comparing the contents of a change package to the contents of a stream, enabling you to easily answer the question, “Have all the changes made for Task A been propagated to Stream B?”

The following sections describe the CLI to the change package facility.

### Adding Entries to a Change Package

The following XML document requests the adding of two entries to the change package of issue record #433: for the element with element-ID 3, record the series of versions between Basis Version 4/3 and Version 4/6; for the element with element-ID 7, record the series of versions between Basis Version 4/2 and Version 4/9.

```
<acRequest>
  <cpkadd>
    <user>jjp</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <issues>
```

```

    <issue>
      <issueNum>433</issueNum>
      <elements>
        <element
          id="3"
          real_version="4/6" basis_version="4/3">
        <element
          id="7"
          real_version="4/9" basis_version="4/2">
        </element>
      </elements>
    </issue>
  </issues>
</cpkadd>
</acRequest>

```

## Removing Entries from a Change Package

The following XML document requests the removal of the change-package entry for the element with element-ID 3 from issue record #433.

```

<acRequest>
  <cpkremove>
    <user>jjp</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <issues>
      <issue>
        <issueNum>433</issueNum>
        <elements>
          <element
            id="3"
            real_version="4/6">
          </element>
        </elements>
      </issue>
    </issues>
  </cpkremove>
</acRequest>

```

## Listing the Contents of a Change Package

The following XML document requests the listing of the change packages of issue records #433 and #512.

```

<acRequest>
  <cpkdescribe>
    <user>jjp</user>

```

```
<depot>etna</depot>
<stream1>etna_dvt</stream1>
<issues>
  <issueNum>433</issueNum>
  <issueNum>512</issueNum>
</issues>
</cpkdescribe>
</acRequest>
```

## Listing Transactions that Affected Change Packages

The following XML document requests the listing of transactions in the range 488–569, annotated with the numbers of the change packages (issue records) modified by those transactions.

```
<acRequest>
  <cpkhist>
    <user>jjp</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <transaction1>569</transaction1>
    <transaction2>488</transaction2>
  </cpkhist>
</acRequest>
```

# The AccuRev Repository

The AccuRev Server program manages a data repository, which provides long-term storage for your organization's development data — for example, all versions of all source files. (Among other things, AccuRev is a special-purpose database management system; the files in the repository — thousands of them — are part of this database.) By default, the repository resides in subdirectory **storage** of the AccuRev installation directory. The repository consists of:

- **site\_slice** directory: implements a database that contains a user registry, list of depots, list of workspaces, and other repository-wide information.
- **depots** directory: contains a set of subdirectories, each storing an individual depot. A depot subdirectory stores one or both of:
  - A version-controlled directory tree: all the versions of a set of files and directories, along with a database that keeps track of the versions.
  - A database of AccuWork issue records.

When it starts, the Server program determines the location of the **site\_slice** directory by looking at the `SITE_SLICE_LOC` setting in configuration file **acserver.cnf**. This file must reside in the same directory as the Server program (**accurev\_server**) itself.

## Repository Access Permissions

The user identity of the AccuRev server process — **acserver** (Unix) or **System** (Windows) — must have full access to all the files and directories within the data repository. For maximum security, this should be the *only* user identity with permission to access the repository. The only exception to this might be an **acadmin** AccuRev administrator account, as suggested in *Unix Systems Only: Administrative User Identities* on page 245.

This user identity must also have access to the **bin** directory where the AccuRev executables are stored.

## READ ME NOW: Assuring the Integrity of the AccuRev Repository

The integrity of the AccuRev data repository is critically important. If information in the repository is lost or corrupted, your organization's ability to do business may be severely compromised. The integrity of the data repository relies on the integrity of underlying software (the file system, including the device drivers for data storage devices) and underlying hardware (the data storage devices themselves). Certain practices will enhance the safety and reliability of these underlying facilities. We strongly recommend the following:

- Use high-quality disk drives and disk controllers.
- Reduce the impact of a hard-disk failure by using disk mirroring (for example, using a RAID system) or other fault-tolerant disk subsystems.

- Power the AccuRev server machine with an uninterruptible power supply (UPS), with automatic shutdown of the server machine if the UPS is running out of power. This reduces the likelihood of interrupted data transfers to disk.
- Establish a good data-backup regimen, and make sure your backups are reliable by doing test restores on a regular basis. (See *Backing Up the Repository* on page 241.)

This section focuses on one aspect of data integrity: guaranteeing “write” operations to the repository. The AccuRev Server process does not, itself, perform the act of writing data on the disk. Like all application programs, it makes a “write” request to the operating system (Unix, Windows). In turn, the operating system performs a “write” operation to the disk itself. (On some larger systems, there may be additional links in this chain of write operations.)

Operating systems and disk subsystems often use special techniques that boost the performance of write operations, but can compromise data integrity. For example, when an application program makes a write request, the operating system might:

- Acknowledge the request immediately — good, because the application program can then proceed to its next operation.
- Delay actually sending the data to the disk (“write-behind”) — bad, because a system failure at this point might result in the data never being stored on the disk.

It is essential that such techniques *not* be used when the AccuRev Server process sends information to the disk containing the AccuRev data repository. The Server always follows each write request with a “synchronize the disk” request. Sometimes, this ensures that data is safely on disk before the Server proceeds to its next task. For example, this is typically the case if the repository is stored on a disk that is local to the machine on which the Server is executing.

But in some situations delayed-write techniques may be used even when the AccuRev Server makes “synchronize the disk” requests. This is typically the case if the repository is located on a network shared file system. In such situations, the Server’s “synchronize the disk” requests are effectively ignored, so that successful completion of write operations to the AccuRev repository cannot be guaranteed. (Some disk subsystems implement such a guarantee by having their own battery backup; buffered data is flushed to disk when the power fails.)

In an attempt to avoid such unsafe situations, the AccuRev Server process attempts to determine whether the file system where the repository is stored guarantees the successful completion of write operations. If it decides “no”, the Server refuses to use the repository. This determination is not foolproof — both “false positives” and “false negatives” are possible.

There’s a workaround in the “false negative” case — where the AccuRev Server process decides that the file system does not guarantee write operations, but *you* know that writes are, in fact, guaranteed. In this case, set environment variable `AC_FS_WRITE_GUARANTEED` to the value 1 in the environment in which the Server process runs; then restart the Server process.

If you have any question about the safety of your data-storage system, please contact AccuRev Support.

## Backing Up the Repository

Note: before you start, consult *A Word of Caution on Windows Zip Utilities* below.

AccuRev supports live backup of the data repository: making copies of the data repository files while the AccuRev Server is running. The **backup** command takes just a few seconds to make checkpoint copies of certain **site\_slice** files in a subdirectory named **backup**. It also records a “high water mark” file, **valid\_sizes\_backup**, in each depot directory, noting the depot’s current transaction level. Transactions that are underway at the time the **backup** command executes are not included.

During **backup** command execution, clients can continue to work, but may notice a slight delay: transactions arriving at the AccuRev Server are queued for execution after completion of the **backup** command.

After executing the **backup** command, you can make a complete copy of the repository (the **storage** directory tree), without worrying about synchronization or time-skew. The append-only nature of AccuRev’s databases makes this simple scheme possible. No matter when you make the backup copies, you’ll be able to restore the repository to its state at the time you executed the **backup** command.

Note: the live-backup scheme relies on the ability to copy files that are currently open to the AccuRev Server process. Your backup utility must be able to copy files that are currently open at the operating system level. If you have any doubts or questions, contact AccuRev support.

Thus, the repository backup procedure is:

1. “Checkpoint” the database portion of the repository:  
`accurev backup mark`
2. If your backup utility cannot copy files that are currently open at the operating system level, stop the **accurev\_server** program. (See *Controlling Server Operation* on page 248.)
3. Use standard operating system backup/restore tools to create a backup copy of the entire directory tree below the **storage** directory. This backup can be all-at-once or piecemeal; for example, you can back up the **site\_slice** directory and the individual subdirectories within the **depots** directory with separate commands.

Good candidates for backup/restore tools are **tar** (Unix), **xcopy /s** (Windows), and **zip** (both).

Note: if your site slice is in a non-standard location (as specified by the **SITE\_SLICE\_LOC** setting in the **acserver.cnf** file — see *Server Configuration File* on page 246), or if some depots are in non-standard locations (perhaps moved with the **chslic** command), then your job in backing up the entire repository is more complicated than simply to copy the **storage** directory.

4. If you stopped the **accurev\_server** program in Step 2, start it again. (See *Controlling Server Operation* on page 248.)



## Restoring the Repository

If you have backed up the repository according to the directions above, you can easily restore the repository to the time at which you executed the **backup** command:

1. Stop the **accurev\_server** program. (See *Controlling Server Operation* on page 248.)
2. Restore the backup copies of the **site\_slice** and individual depot directories, using the standard backup/restore tools.
3. Go to the **site\_slice** directory.
4. Overwrite database files with the checkpoint files in the **backup** subdirectory:

```
cp backup/* . (Unix)
```

```
copy backup\*. * . (Windows)
```

5. Reindex the site slice:

```
maintain reindex
```

6. Restore and reindex each depot:

```
maintain restore <depot-name>
```

```
maintain reindex <depot-name>
```

7. Restart the **accurev\_server** program. (See *Controlling Server Operation* on page 248.)

Note: suppose a particular depot's files were not backed up for several hours after the **backup** command was executed. During that interval, several new versions of file **gizmo.c** were created with the **keep** command. All of those versions will officially be lost when the repository is restored to its state at the time the **backup** command was executed. But you can still retrieve a copy of the last-created lost version of file **gizmo.c** from the backup medium.

## Archiving Portions of the Repository

The container files that store the contents of individual file versions can now be move to offline storage, in order to save online storage space for the repository. For details, see *Archiving of Version Container Files* on page 255.

## Moving a Workspace or Reference Tree

Note: before you start, consult *A Word of Caution on Windows Zip Utilities* below.

First, move the physical contents of the workspace tree or reference tree with standard operating system tools (e.g. **tar**, **zip**, **xcopy** /s). Then, let AccuRev know about the move:

```
accurev chws -w <workspace-name> -l <new-location>
```

```
accurev chref -r <reftree-name> -l <new-location>
```

## Moving a Depot

Note: before you start, consult *A Word of Caution on Windows Zip Utilities* below.

First, move the physical contents of the depot with standard operating system tools (e.g. **tar**, **zip**, **xcopy** /s). Then, let AccuRev know about the move with this command:

```
accurev chslice -s <slice-number> -l <new-location>
```

(Use **accurev show depots** to determine the slice number of the depot.)

## Removing a Depot

A depot can be removed completely from the repository with the **maintain rmdepot** command. This operation is irreversible! For details, see *Removing a Depot from the AccuRev Repository* on page 296.

## A Word of Caution on Windows Zip Utilities

Be careful when using WinZip® or PKZIP® on a Windows machine to perform the tasks described above: backup/restore of the entire repository, or moving a workspace, reference tree, or depot. You may want to use **tar** on a Unix machine to “pack up” a directory tree, and then use the Zip utility on a Windows machine to “unpack” it.

- When moving the entire repository or an individual depot, be sure to disable conversion of line-terminators during the “unpack” step:
  - In WinZip, make sure the option “TAR file smart CR/LF conversion” is not selected (**Options > Configuration > Miscellaneous**).
  - In PKZIP, make sure the “CR/LF conversion” setting is “None -- No conversion” (**Options > Extract**).

Enabling conversion of line-terminators during the “unpack” step will corrupt the text files in a depot's file storage area (see *File Storage Area* below). The AccuRev Server always expects lines in these text files to be terminated with a single LF character, no matter what kind of machine the server is running on.

- Conversely, when moving a workspace or reference tree, you may wish to enable “TAR file smart CR/LF conversion”. The files in a workspace or reference tree are handled directly by text-editors, compilers, testing tools, etc. Many Windows text-editors are incapable of handling text files whose lines are terminated with a single LF character.

## Storage Layout

Each AccuRev depot is stored in a separate directory tree under the installation area's **storage** directory. The **storage** directory is a sibling of the executables (“bin”) directory. For example, if AccuRev is installed at **/usr/accurev** and depots named **moe**, **larry**, and **curly** are created, the

directory layout would be:

```
/usr/accurev
  bin
  storage
    site_slice
    depots
      moe
      larry
      curly
```

A depot consists of three parts:

### Configuration Files

The **mktrig** command creates a one-line configuration file that names the script to be executed when the trigger fires for transactions involving this particular depot. For example, making a trigger of type “pre-keep-trig” creates a configuration file in the depot named **pre-keep-trig**. (This file might contain the pathname **/usr/local/bin/accurev\_prekeep.pl**.)

### Metadata Area

The metadata area stores information about versions, times, and source file storage locations within the source file storage area of the depot. The metadata is stored in files ending with **.ndb** and **.ndx**.

The metadata area must be physically located on the machine where **accurev\_server** is running. This guarantees the integrity of physical disk writes. Moving the metadata area to a remote file system compromises data integrity and is not supported by AccuRev, Inc.

### File Storage Area

Whenever a user creates a new real version of a file with the **keep** command, the AccuRev Server copies the file from the user’s workspace to the depot’s file storage area. The newly created storage file is permanently associated with the real version-ID in the workspace stream (e.g. 25/13), and also with subsequently created virtual version-IDs in higher-level streams (7/4, 3/9, 1/4).

Storage files are located in subdirectory tree **data** within the depot directory. The files may be in compressed or uncompressed form. Compressed files may correspond to more than one real version. Conceptually, storage files are numbered sequentially starting with 1 and going up to  $2^{**64}$ . (That should be enough.) Within the **data** directory, they’re arranged in a hierarchy for faster access. For example, storage file #123456 would be stored as **data/12/34/56.sto**.

Recovery information for the storage file is stored in a like-named **.rrf** file (e.g. **data/12/34/56.rrf**).

You can relocate a depot’s file storage area onto other disk partitions or even onto remote disks. The cautions about storing data locally do not apply to files in the data directories. However, exercise extreme caution when relocating storage in this area. Make sure you have first done a full backup and have shut down the **accurev\_server** program.

# The AccuRev Server

The AccuRev data repository is managed by a single program, the AccuRev Server (**accurev\_server**). This program must be started prior to running any AccuRev client commands. The server program should be the only process that directly manipulates the AccuRev repository. No person should attempt to work directly with the repository, unless it is an emergency.

## User Identity of the Server Process

The AccuRev Server process, named **accurev\_server**, has a user identity at the operating system level. This process should run with a special user identity, to help ensure that no other user or process modifies the data repository:

- Unix: create a user named **acserver**.

You may be tempted to simply let the AccuRev Server process run as **root**. But we strongly recommend against this, as it would open a large security hole. The Server can run user-supplied trigger scripts (see *AccuRev Triggers* on page 275). In general, having user-supplied scripts run as the **root** user is very dangerous!

- Windows: don't create a new user; the AccuRev Server process runs as the built-in local user named **System**.

This user identity must have access to the AccuRev executables (**bin**) directory and to the data repository (see *Repository Access Permissions* on page 239).

Note: **acserver** and **System** are user accounts at the operating system level. You don't need to — and should not — create a principal-name (AccuRev user name) “acserver” or “System”. On the other hand, the AccuRev Server might need to take on an AccuRev user identity when it executes a server-side trigger script. For more on this topic, see *Trigger Script Execution and User Identities* on page 290.

## Unix Systems Only: Administrative User Identities

You can control the user identity under which the Server runs with the server configuration file. See *Server Configuration File* on page 246.

Not even Unix administrators should run as **acserver** unless it is an emergency. If you want to have an AccuRev administrator account, we recommend creating a separate account named **acadmin**. Place both **acserver** and **acadmin** in a group named **acgroup**, and record these names in the server configuration file:

```
USER = acserver
GROUP = acgroup
```

With this setup, the **acadmin** user will be able to access the repository. You can configure Unix-level auditing and place other appropriate controls on this account; this leaves the **acserver** account (and thus, the AccuRev Server process) unencumbered by such controls.

## Starting the AccuRev Server

The following sections describe how to start the AccuRev Server program, either automatically at operating system bootstrap time, or manually at a command prompt. (You can also perform a “manual” startup with a Unix shell script or a Windows batch file.)

### Running the Server Automatically at Operating System Startup

Typically, the Server program is started automatically when the operating system boots on the server machine. On Unix systems, an “rc” or “init.d” startup script starts the **accurev\_server** program. The AccuRev installation program does not install the startup script automatically, however. You must customize and install the sample startup script located in the **extras/unix\_init** subdirectory of the AccuRev installation directory. See the **README** file for complete instructions.

On Windows NT/2000/XP systems, the AccuRev installation program automatically configures the **accurev\_server.exe** program as a Windows service. Use the standard **Services** applet on the Windows Control Panel to control the Server program.

### Starting the Server Manually

The AccuRev Server must be started manually in the following environments:

#### On Windows 95/98/ME systems

Start the Server with the **AccuRev Demo Server** shortcut on the desktop. Alternatively, run the **server\_start.bat** script, located in the AccuRev executables (**bin**) directory.

#### On Windows NT/2000/XP systems

If you’ve changed the startup type of the AccuRev service to “Manual”, you can start the service from the **Services** applet. Alternatively, run the **server\_start.bat** script, located in the AccuRev executables (**bin**) directory.

#### On Unix systems

Start the Server with the **acserverctl** utility:

```
<AccuRev-executables-dir>/acserverctl start
```

## Server Configuration File

When it starts, the Server program reads configuration file **acserver.cnf**, located in the AccuRev executables directory. This configuration file is generated during installation, but can be edited manually thereafter.

Here is a sample **acserver.cnf**:

```
MASTER_SERVER = accurev_server_machine.company.com
SITE_SLICE_LOC = /partition0/site_slice
```

**IMPORTANT NOTE:** the white space surrounding the equals sign (=) in configuration files is mandatory.

The name of the server machine should be the fully-qualified server name, including a domain name and Internet extension. Using just the server name may work in most situations, but fully-qualified is preferred. Alternatively, you can use the IP address of the server machine.

The `SITE_SLICE_LOC` setting points to the directory that the server uses for storing information about the site such as the user registry, depot information, etc. This directory should be owned by the **acserver** account (Unix) or the System account (Windows) and must be physically located on the server machine. The `SITE_SLICE_LOC` location must not be within a remotely mounted file system (Unix) or within a shared directory (Windows) on a remote machine.

## Unix Systems Only: Controlling the Server's User Identity

Note: this section applies to Unix machines only. On Windows machines, the user identity of the AccuRev Server is always the local **System** account.

If you perform a server installation as the **root** user, the following specifications are stored in the **acserver.cnf** file:

```
USER = nobody
GROUP = nobody
```

When the Server starts automatically at system bootstrap time, it reads these specifications and assumes the identity: user **nobody** in group **nobody**.

You can edit these settings to change the AccuRev Server's user/group identity. This change takes effect the next time the Server is started by the **root** user — either automatically at system bootstrap time or manually, using the **acserverctl** utility. (See *Controlling Server Operation* below.)

When the Server is started manually, it runs under the identity of the user who started it.

## Server Watchdog

The AccuRev Server is designed for high reliability, to ensure the integrity of the repository and its availability to AccuRev users. But even the robust software systems are occasionally compromised; the AccuRev Server can be brought down by a bad disk sector or an administrator's mistaken command.

The reliability of the AccuRev Server is further enhanced by a companion program, termed the Watchdog, which runs on the same machine. The sole function of the Watchdog is to monitor the Server and restart it in the event of a failure. The effect of the Watchdog on Server performance is insignificant.

Note: both the Server and Watchdog show up in the operating system's process table with the same name: **accurev\_server**.

For the most part, the Watchdog is "transparent", making administration simple:

- The Watchdog process starts automatically when the Server process is started (typically, at operating system bootstrap time).
- The administrative commands for stopping the Server process cause both the Watchdog and Server to stop. These commands have been reworked to terminate the Watchdog directly; before it exits, the Watchdog terminates the Server.

Tools that control the execution of the Server and Watchdog are in described in section *Controlling Server Operation* on page 248.

## Server Logging

The AccuRev Server maintains a log file in subdirectory **logs** of the **site\_slice** directory:

- The name of the log file is **acserver.log**. (In previous releases, the log file was named **accurev\_server.log** and was located in the **site\_slice** directory itself.)
- Each log message includes a timestamp, a client-server connection ID, the user's principal-name, the name of the server subtask being performed, the ID of the thread performing the server subtask, and the IP address of the client machine. For example:  
  

```
2003/05/09 12:30:36    1002 jjp cur_wspace    0x803 127.0.0.1
```
- Log file rotation” keeps the log file from growing too large. Periodically, the AccuRev Server timestamps the current log file and moves it to subdirectory **logs** of the **site\_slice** directory. For example, the log file might be renamed **acserver-2002-01-23-04-47-29.log**. The Server then creates a new **acserver.log** file.

The log file is rotated weekly; it is also rotated whenever the AccuRev Server is restarted.

## Watchdog Logging

The Watchdog also maintains a simple log file, **acwatchdog.log**, in the **logs** directory. The Watchdog log file is rotated in the same way as the Server log file.

## Controlling Server Operation

AccuRev includes facilities for controlling the operation of the AccuRev Server and the new Watchdog. The user interface varies by platform:

- Unix: a special command-line utility
- Windows: the standard **Services** console

### Unix: ‘acserverctl’ Utility

If the AccuRev Server is running on a Unix machine, you can control its operation with the **acserverctl** program. This is a Bourne-shell script, located in the AccuRev **bin** directory. (It is based on the control script for the Apache Web server.)

Note: by default, **acserverctl** assumes that AccuRev is installed at **/opt/accurev**. If this is not the case, you must run **acserverctl** in an environment where **ACCUREV\_BIN** is set to the pathname of the AccuRev **bin** directory. For example:

```
env ACCUREV_BIN=/var/accurev/bin acserverctl ...
```

**acserverctl** provides a set of non-interactive commands. The format of each command is:

```
acserverctl <command-name>
```

(Omitting *<command-name>* is equivalent to executing **acserverctl help**.) The commands are:

### **start**

Start the Server and Watchdog processes.

### **stop**

Tell the Server and Watchdog processes to stop gracefully.

### **status**

Report whether the Server is running or not.

### **pause**

Tell the Server to stop accepting new requests from AccuRev clients.

### **resume**

Tell the Server to start accepting new requests from AccuRev clients again.

### **restart**

Tell the Server process to stop gracefully; this allows the Watchdog to restart it. If the Watchdog is not running, a **start** or **hardrestart** is performed.

### **kill**

Forcibly stop the Server and Watchdog processes. This is accomplished by sending a TERM signal to each process. The script gets the process-IDs from files **acserver.pid** and **acwatchdog.pid**, located in the **site\_slice** directory. These files are written automatically when the processes are started.

### **hardrestart**

Perform a **kill**, followed by a **start**.

### **help**

Display an **acserverctl** command summary.

The various “tell a process” capabilities are implemented through server-control files. (See *Server-Control Files* below.)



## Windows: ‘Services’ Console

If the AccuRev Server is running on a Windows machine, you can control its operation from the standard Windows **Services** console. The **Services** console is located in the Windows **Control Panel**; in some versions of Windows, it’s in a subfolder called **Administrative Tools**.

The context (right-click) menu of the AccuRev service includes these commands:

- start**
- stop**
- pause**
- resume**
- restart**

For descriptions of these commands, see *Unix: ‘acserverctl’ Utility* above. On Windows, the **restart** command brings down both the Server and the Watchdog, by performing a **stop** followed by a **start**.

## Server-Control Files

On all platforms, the AccuRev Server and Watchdog processes check, once per second, for the existence of several “server-control files” in the **site\_slice** directory. The existence of the server-control file causes the process to perform a particular action. In most cases, the contents of the file are irrelevant; a zero-length file will do.

### **acserver.command.pause**

(used by the **pause** server-control command) Tells the Server to stop accepting new requests from AccuRev clients. The Server completes transactions that are already in progress and logs its “paused” status to the log file. Then, it continues to run, but the only thing it does is monitor the **acserver.command.pause** file. When this server-control file is removed, the Server resumes normal operation.

This server-control file is *not* removed when a new Server process starts up. If the file exists, the Server starts up in the paused state.

### **acserver.command.shutdown**

(used by the **stop** and **restart** server-control commands) Tells the Server to “finish up quickly” and exit. The Server immediately stops accepting new requests from AccuRev clients. It continues to work on transactions that are already in progress, but it aborts any transactions that are not completed within 10 seconds. Then, the Server exits.

If the Watchdog is running, it detects the Server’s shutdown and starts up a new Server immediately. Thus, this server-control file typically causes a Server restart. In any event, this file is automatically removed whenever a new Server process starts up.

If 10 seconds is not the appropriate interval for “finishing up quickly”, place another integer (such as 120) in the **acserver.command.shutdown** file. The Server exits when there are no more transactions to work on, or when the timeout interval has passed, whichever comes first.

## **acwatchdog.command.shutdown**

(used by the **stop** server-control command) Tells the Watchdog to exit cleanly. When the Server detects that the Watchdog has exited, it exits also, just as if it had found an **acserver.command.shutdown** file (see above). In this case, however, there is no longer a Watchdog process, so no restart of the Server takes place.

This server-control file is automatically removed when a new Server process starts up.

## **Open Filehandle Limits and the AccuRev Server**

The AccuRev Server is designed to handle multiple client commands concurrently: any number of requests that “read” data, along with one command that “writes” data. Accomplishing such concurrency typically requires that the AccuRev Server have many files open at the same time. Each operating system imposes limits on how many files can be open simultaneously. There may be an “open file descriptor” limit for each user process, or an overall limit for all user processes, or both. If the AccuRev Server hits the open file descriptor limit, additional client requests will be queued until file descriptors become available. (No client command is cancelled, and no data is lost. Hitting the open file descriptor limit just slows AccuRev Server performance.)

The table below indicates the default open file descriptor limits for the various AccuRev-supported operating systems. Following the table are instructions for increasing these limits.

<b>Operating System</b>	<b>Default Limit on Open Filehandles</b>	<b>Command to Change Limit on Open Filehandles</b>
Windows NT, Windows 2000, Windows 2003, Windows XP Pro	2048 per system	none
Windows 98, Windows ME, Windows XP Home	2048 per system	none
Solaris 7	64 per process	no command; must reconfigure Unix kernel (see below)
Linux (Red Hat Fedora 3)	about 3500 per system	/sbin/sysctl -w fs.file-max=10000
Linux (Power PC 2.4)	1024 per process	
AIX 5.1	2000 per process	
HP-UX 11	360 per process	no command; must rebuild Unix kernel (see below)
Tru64 5.1	4096 per process	
IRIX 6.2	1024 per process	
SCO UnixWare 7.1.1, 7.1.4	60 per process	no command; must rebuild Unix kernel (see below)

Note: If you are performing a pre-purchase evaluation of AccuRev in an environment with a limited number of users and a limited amount of data, there is no need to make any changes. The default limits will be more than adequate.

## Changing the Per-Process Open File Descriptor Limit

The procedure for increasing a process's maximum number of open files varies from operating system to operating system.

Note: in all cases, be sure to remove file **acserver.handle.limit**, located in the AccuRev **site\_slice** directory, before restarting the AccuRev Server or rebooting the operating system. This file caches the current value of the open-files limit.

### Linux

You must be the **root** user to perform the following procedure.

1. Change the overall limit on the number of open file descriptors each process can have (e.g. to 10,000):

```
> /sbin/sysctl -w fs.file-max=10000
```

The number you specify is stored in file **/proc/sys/fs/file-max**.

2. Add this line to file **/etc/pam.d/login**:

```
session    required    /lib/security/pam_limits.so
```

3. Change the capabilities of the Bash shell command **ulimit**, by creating or editing the “nofile” (number of open files) lines in file **/etc/security/limits.conf**. Example:

```
*    soft    nofile    1024
*    hard    nofile    10000
```

These lines specify that:

- By default, a Bash shell (and its subprocesses) can have as many as 1024 open file descriptors.
- A Bash shell can execute the command **ulimit -n *number***, with 65535 as the maximum value of ***number***. This enables that particular shell (and its subprocesses) to have up to ***number*** open files. (The person executing the **ulimit** command doesn't need to know what the “hard limit” specified in **/etc/security/limits.conf** is — he can just enter the command as **ulimit -n unlimited** to get the maximum value.)

4. Test your change, by entering a **ulimit** command in a Bash shell, setting the limit somewhere in between the **soft** and **hard** specifications you made in Step 3. For example:

```
> /bin/bash

$ ulimit -n
1024

$ ulimit -n 5000

$ ulimit -n
5000
```

5. Restart the AccuRev Server process from a Bash shell:

```
> <AccuRev-bin-directory>/acserverctl stop(stop the AccuRev Server)

> /bin/bash(start a Bash shell)

$ <AccuRev-bin-directory>/acserverctl start(restart the AccuRev Server)
```

## Solaris

You must be the **root** user to perform the following procedure.

1. Change the overall limit on the number of open file descriptors each process can have (e.g. to 5,000), by adding or changing this line in file **/etc/system**:

```
set rlim_fd_max=5000
```

2. Reboot the operating system.

## HP-UX

You must be the **root** user to perform the following procedure.

1. Enter this command to start the System Administration Manager utility:

```
> sam
```

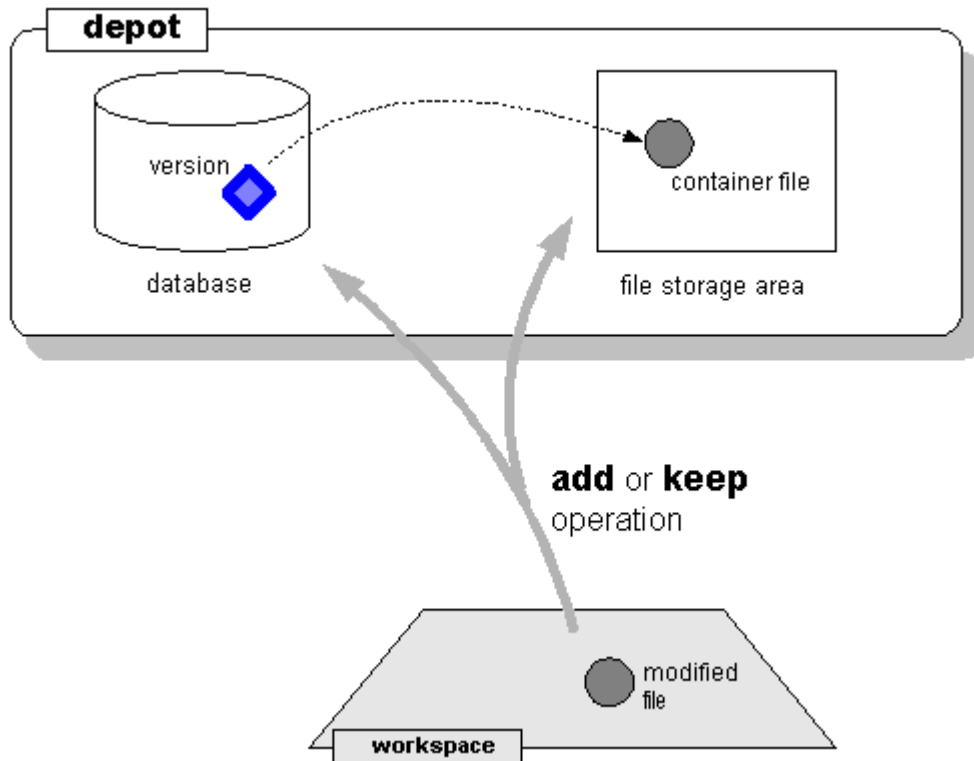
2. Select **Kernel Configuration**, then **Configurable Parameters**.
3. Select the **maxusers** parameter.
4. Increase the value of this parameter, for example to 128.
5. Invoke the **Actions > Process New Kernel** command, to create a new HP-UX kernel.
6. Exit the System Administration Manager utility.
7. Reboot the operating system.



## Archiving of Version Container Files

Users execute a **keep** command to preserve the current contents of a version-controlled file (“file element”) in an AccuRev depot. Similarly, users execute an **add** command to place a file under version control. The **add** and **keep** commands:

- copy the current contents of the file to a container file, located in the depot’s file storage area.
- create an associated version object in the depot’s database.



In accordance with the TimeSafe principle, the version object can never be deleted from the database or modified in any way. The corresponding container file is always accounted for, and can be in either of these states:

- **normal** — the container file is located in the depot’s file storage area (the **data** subdirectory of the depot directory). AccuRev commands, such as **update**, **cat**, and **diff**, can access the contents of the version.
- **archived** — the container file has been moved to a gateway area outside the depot’s file storage area. AccuRev commands cannot access the contents of an archived version. After container files have been moved to the gateway area, an administrator can use standard operating system or third-party tools to transfer the container files to off-line storage: tape, CD-ROM, removable disk drive, Web-accessible storage, etc.

Only binary files can be in the archived state. Text files are always in the normal state.

The AccuRev CLI commands **archive** and **unarchive** shift container files back and forth between the normal and archived states. Before using **unarchive**, the administrator would transfer the appropriate container files from off-line storage back to the gateway area. Then, invoking **unarchive** moves the container files back into the depot's **data** directory.

## The 'archive' Command

The command **accurev archive** processes one or more versions of file elements, shifting the versions' container files from **normal** status to **archived** status. The command has this format:

```
accurev archive [ -i ] [ -s <stream> ] [ -t <transaction-range> ]  
[ -c <comment> ] [ -R ] [ -E <elem-type> ] <element-list>
```

## Determining Which Versions to Archive

**archive** determines the set of versions to archive as follows:

- It focuses on a particular stream. If you don't specify a stream with **-s <stream>**, it uses your current workspace's stream.
- It narrows the scope to a particular set of file elements, which you specify as command-line arguments in the **<element-list>**. You can include directories in this list; in this case, use the **-R** option to include the recursive contents of those directories.
- By default, all eligible versions of the specified elements in the specified stream are archived. You can use **-t** to limit the set to the versions of those elements created in a specified transaction, or range of transactions:

**-t <number>** *single transaction*

**-t <number>—<number>** *range of transactions*

- You can also limit the set of versions to those of a particular element type, using the **-E** option.

Note: you *must* specify the **binary** element type: **-E binary**.

The **archive** command refuses to archive any version that is currently visible in any stream or snapshot. It also refuses to process versions in text format. Only binary-format versions will be archived.

Using the **-i** option (in addition to the other options described above) generates an XML-format listing of the desired versions, but does not perform any actual archiving work. It is highly recommended that you do this before actually archiving any versions.

## Archiving the Versions

After determining which versions to process, the **archive** command moves a version's container file from a "normal" location under the **data** directory:

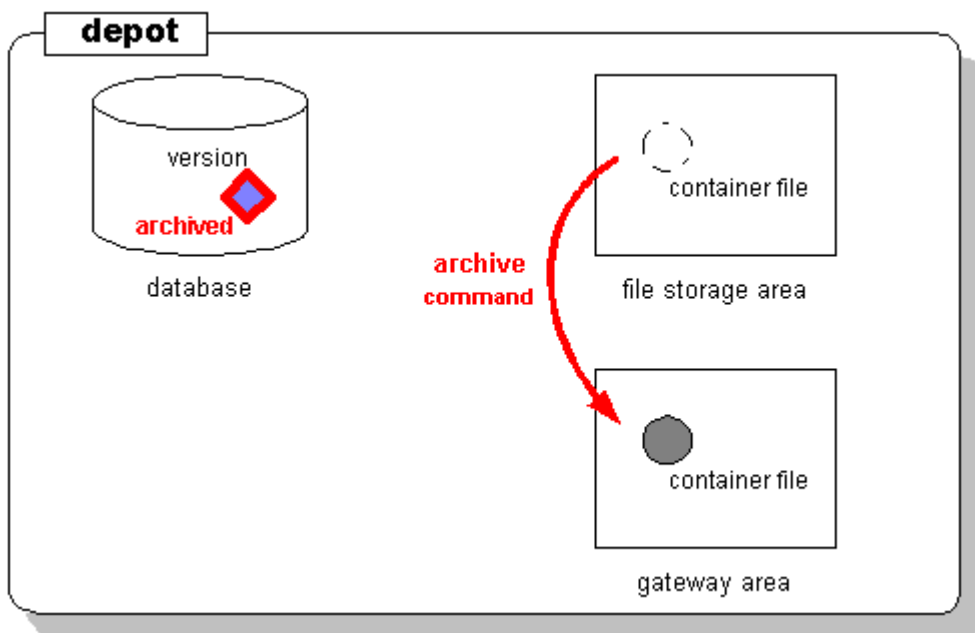
```
.../storage/depots/gizmo/data/25/07.sto
```

... to a corresponding “archived” location in the **archive\_gateway/out** area:

```
.../storage/depots/gizmo/archive_gateway/out/data/25/07.sto
```

**archive** also marks the version as “archived” in the depot database.

Subsequent attempts by AccuRev commands to retrieve the contents of the archived version will fail.



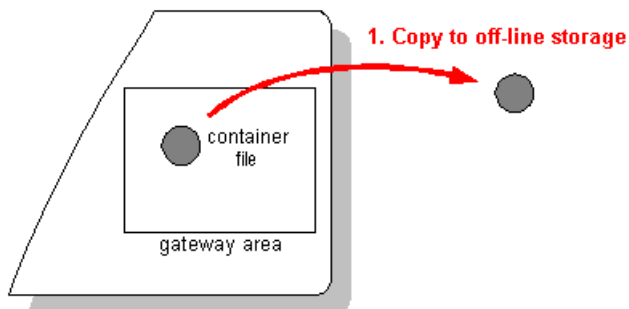
The changes made by this command are recorded in the depot database as a transaction of type **archive**. You can use the **-c** option to specify a comment string to be stored in this transaction. You can search for particular comment strings when using the **hist** command to locate previous **archive** transactions. See [Using 'hist' to Research Previous 'archive' Commands](#) on page 258.

## The ‘reclaim’ Command

The **archive** command merely moves container files from one location (the depot’s **data** area) to another location (the depot’s **archive\_gateway** area). To reduce the amount of disk space consumed by the archived versions, you must:

1. Copy the files from the **archive\_gateway** directory tree to off-line storage. You can use operating system commands (**copy**, **xcopy**, **cp**, **tar**) and/or third-party data-backup utilities to accomplish this.

Be sure to use a tool that preserves the source data’s directory hierarchy in the copied data.



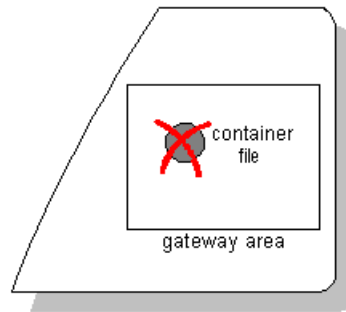


**WARNING!** AccuRev has no way of tracking which tool you use for this purpose, or what off-line storage medium you copy the files to. It's up to you to maintain good records of these activities!

2. Delete the files from the **archive\_gateway** directory tree, using the **reclaim** command:

```
accurev reclaim [ -p <depot> ]  
               -t <archive-transaction>
```

You must specify a single transaction of type **archive**, created by previous **archive** command(s).



2. Reclaim disk space

## Attempts to Access Archived Versions

The **archive** command affects depot storage only. It has no immediate effect on any workspace. But you might subsequently enter an AccuRev command that attempts to access a version that has been archived. For example, if version **gizmo\_int/8** of file **floor\_layout.gif** has been archived, then this command fails:

```
accurev cat -v gizmo_int/8 floor_layout.gif > old_layout.gif
```

In such cases, a message is sent to **stderr** and the command's exit status is 1.

## Using 'hist' to Research Previous 'archive' Commands

Each depot's database contains a complete record of all version-archiving activity for that depot. Execution of the **archive** command is recorded as a transaction of kind **archive**. You can use the **hist** command to locate all such transactions:

```
accurev hist -a -k archive
```

You can also select just those **archive** transactions that were created with a particular comment string:

```
accurev hist -a -k archive -c "stadium images"
```

In a **reclaim** command, you must indicate the storage space to be reclaimed by specifying the number of an **archive** transaction.

## Restoring Archived Versions — The 'unarchive' Command

After you have **archived** some versions and **reclaimed** the disk space:

- the versions' container files are no longer in the depot's **data** area.
- copies of the container files are no longer in the depot's **archive\_gateway/out** area (since you've transferred them to off-line storage).

If you decide you need to restore some or all of the archived versions, you must first copy the container files from off-line storage back to the **archive\_gateway** area. You must place the files under **archive\_gateway/in**, at the same relative pathname as they were originally placed under **archive\_gateway/out**. For example, if the **archive** command places a container file at:

```
.../storage/depots/gizmo/archive_gateway/out/data/25/07.sto
```

... you must restore the file from off-line storage to this pathname:

```
.../storage/depots/gizmo/archive_gateway/in/data/25/07.sto
```

After placing all the container files in the **archive\_gateway/in** area, you can execute the **unarchive** command. This command has exactly the same format as **archive** — that is, you specify the versions to be restored in exactly the same way as you originally archived them. For example:

Archive all non-active versions of GIF image files in stream **gizmo\_maint\_4.3**:

```
accurev archive -s gizmo_maint_4.3 -E binary *.gif
```

Restore all those versions:

```
accurev unarchive -s gizmo_maint_4.3 *.gif
```



# Replication of the AccuRev Repository

This chapter describes how to set up and use AccuRev's repository replication feature. One server machine stores the “master” copy of the AccuRev data repository; any number of additional server machines can store “replicas” of the repository. Each replica contains some or all of the repository's depots. Users can send commands to the AccuRev Server software running on any of these machines.

Note: use of the repository replication feature requires purchase of the *AccuRev Replication Server* product from AccuRev, Inc.

## Master and Replica

One host in the network acts as the AccuRev server machine: it runs the AccuRev Server process and contains the AccuRev repository in its local disk storage. In a replication scenario, this original host is termed the master server.

One or more additional hosts in the network can act as replica servers. Each such host runs its own instance of the AccuRev Server process; likewise, each such host has its own copy of the AccuRev repository. The diagram below shows the servers in a replication scenario, along with various client machines.

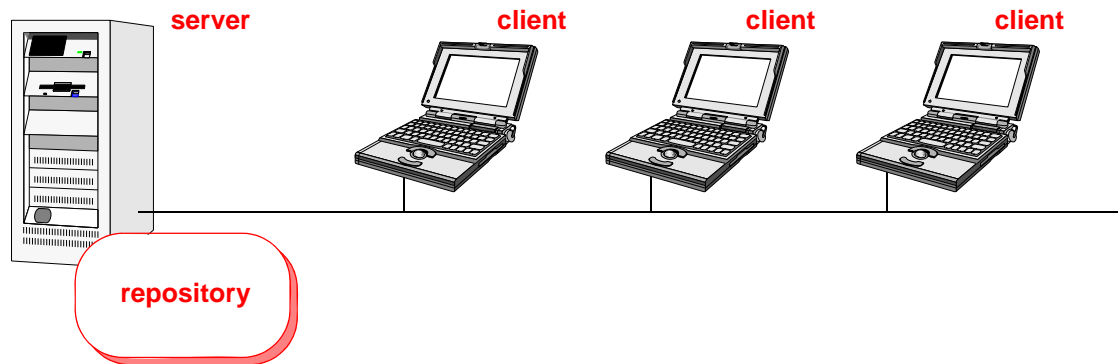
We use the terms master repository and replica repository to distinguish the multiple repositories in a replication scenario. The master repository is always complete and up-to-date; all transactions (operations that change the repository) are handled by the master server and are logged in the master repository.

By contrast, a replica repository can become out of date during day-to-day usage: it can be missing recent transactions initiated by clients using other replica servers or the master server. You can issue a simple synchronization command to download such missing transactions from the master repository to the replica repository. This makes the replica repository database into an exact copy (temporarily, at least) of the master repository database. Synchronization also occurs automatically whenever a transaction is initiated by a client using that replica server.

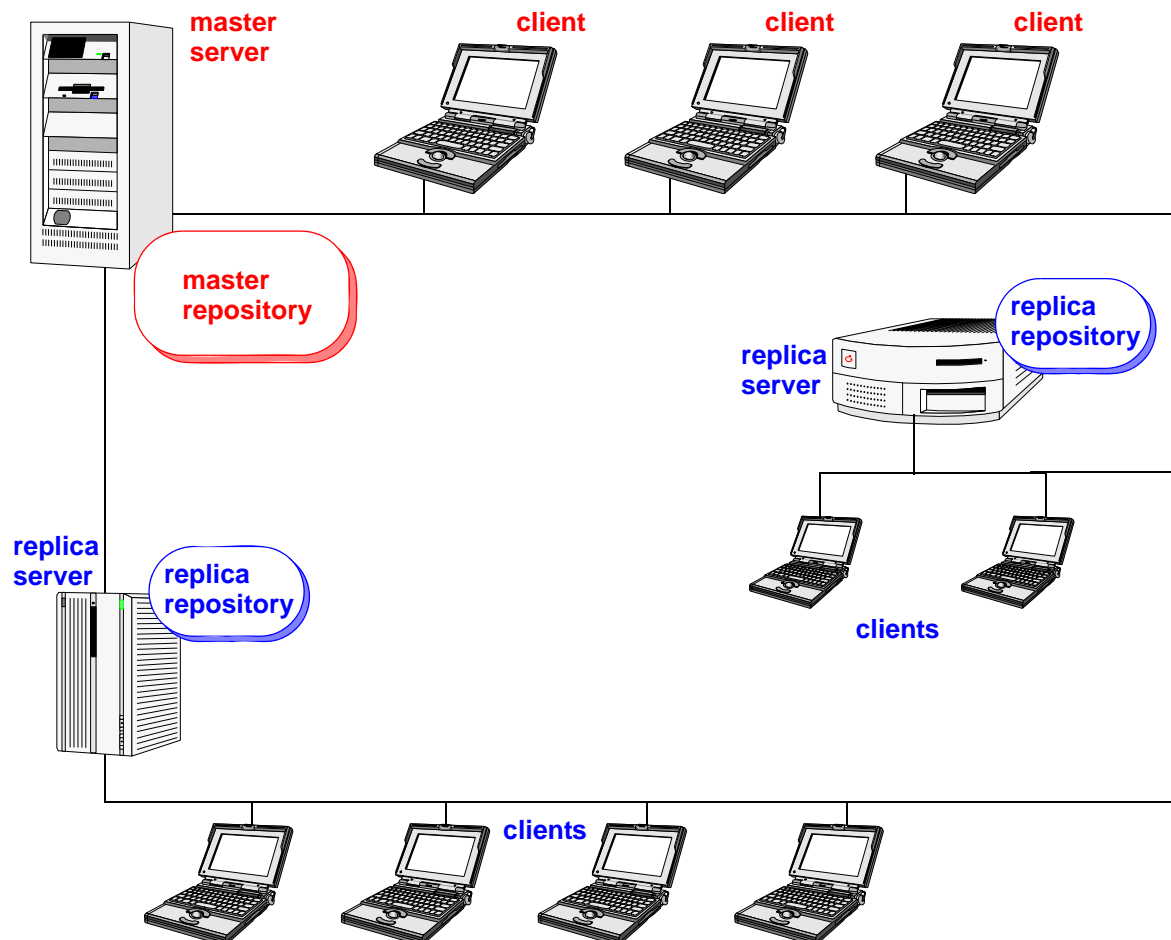
A replica repository can contain a selected subset of the depots in the master repository. If the master repository contains 10 depots, one replica repository might be configured to contain 4 of the depots, another replica repository might be configured to contain 7 of them, and a third replica repository might be configured to contain all 10 depots.

For more details on day-to-day operations involving master and replica repositories, see the sections starting with *Using a Replica Server* on page 266. First, we address licensing issues and describe the replica setup process.

## Before Replication



## After Replication



## AccuRev Licensing in a Replication Environment

A standard AccuRev license (Professional Edition or Enterprise Edition) enables your organization to maintain a single data repository, managed by a single AccuRev Server process running on a particular machine. This license also enables a certain number of users to access the repository with AccuRev client software.

The most obvious (and easiest) way to add replication to this environment is to have the existing server machine become the master server. Let's say that you want to replicate the repository at two remote sites — one with 10 client users, the other with 25 client users. In this case, your organization needs to purchase:

- two copies of AccuRev Replication Server
- ten individual Remote User Licenses
- one Remote Site License (for the 25 users at the second remote site)

## Installation Procedure: Assumptions

The procedures in the following sections make the following assumptions about your AccuRev installation:

- The original AccuRev server machine is named **masthost**. If one or more replicas have already been created, then this machine is already the master server.
- The machine to be made into a replica server is named **replhost**.

Restriction: **masthost** and **replhost** must both have “big-endian” hardware architectures, or must both be “little-endian”. Little-endian architectures include Intel and AMD, running either Windows or Linux software. Big-endian architectures include Sparc (Sun) and PA-RISC (Hewlett-Packard), and PowerPC (AIX).

The same host can act both as the master server and as the replica server (or even as multiple replica servers). This can be convenient for performing testing of new releases, validating your organization's development procedures, etc. Before proceeding to the next section, read the notes in *Using the Same Host as Both Master Server and Replica Server* on page 266.

## Setting Up the Master Server

The change described in this section needs to be made just once — when transitioning from a non-replicated setup to a replicated setup.

1. Stop the AccuRev Server process on **masthost**.
2. Edit the **acserver.cnf** file, which is located in the AccuRev **bin** directory. Add the following line:

```
REPLICATION_ENABLED = true
```

CAUTION: enabling replication poses a potential security risk. Before proceeding, be sure to read *Synchronization Security* on page 269.

3. Note the MASTER\_SERVER and PORT settings in the **acserver.cnf** file. You'll need these settings in Step 8 below.
4. Restart the AccuRev Server process on **masthost**.

## Setting Up the Replica Server

The following sections detail the steps for setting up **replhost** as an AccuRev replica server, using the repository data from **masthost**.

### Install AccuRev

5. Obtain from AccuRev Customer Support a **keys.txt** file containing a license to run the AccuRev Server software on **replhost**. Most specifications, including the number of users, should match the license on **masthost**. The host name will differ — and the port number might differ, too.
6. Install the AccuRev software on **replhost**. During installation, choose both the Custom and Full options. If you've already installed AccuRev on this machine, choose a new location for the installation directory. The installation wizard will prompt you to specify the storage location for the new, local repository on **replhost** ("Customize: Choose a Folder for AccuRev Server Data Storage").

In the following steps, we'll assume that the repository location is **/opt/accurev/storage** — adjust for your installation and operating system.

### Revise the Server Configuration File

7. Stop the AccuRev Server process on **replhost**.
8. Edit the **acserver.cnf** file, which is located in the AccuRev **bin** directory.
  - Change the keyword MASTER\_SERVER to LOCAL\_SERVER, and change the keyword PORT to LOCAL\_PORT. But don't change the value of either setting.
  - Add new MASTER\_SERVER and PORT settings, using the values of these settings on **masthost**. (These are the settings you noted in Step 3.)

After these edits, the four lines might look like this:

```
MASTER_SERVER = masthost
PORT = 5050
...
LOCAL_SERVER = replhost
LOCAL_PORT = 5050
```

Note: there is no relationship between the LOCAL\_PORT and PORT numbers. They can be the same or different.

## Start the AccuRev Server Process

9. Make sure that the **replhost** operating system process in which the AccuRev Server will run has an AccuRev username (“principal-name”) identity with enough rights to access all the files in the **masthost** repository. AccuRev ACLs control access to depots and streams for specified AccuRev users and groups.

- Unix: use environment variable ACCUREV\_PRINCIPAL to establish the AccuRev principal-name of the operating system process. For example, in a Unix Bourne shell:

```
ACCUREV_PRINCIPAL=acadmin; export ACCUREV_PRINCIPAL
```

- Windows: the AccuRev Server runs as a Windows service; reconfigure it to run as the desired AccuRev principal-name, instead of LocalSystem. In the Control Panel’s Services applet: open the Properties window for the AccuRev service, go to the Log On tab, select “This account”, and enter the AccuRev principal-name and Windows password.

On either Unix or Windows, make sure that the AccuRev-level password of the desired principal-name is set. (This password is independent of OS passwords.) The AccuRev-level password is stored in file **authn**, in the **.accurev** subdirectory of *principal-name*’s home directory.

10. Start the AccuRev Server process on **replhost**.

## Synchronize the Site Slice

11. Run this client command on **replhost**:

```
accurev replica sync
```

This command copies data from **masthost**’s site slice to **replhost**’s site slice. In particular, it makes the AccuRev Server on **replhost** aware of all the depots in the master repository on **masthost**.

## Configure the Replica Server to Include the Desired Depots

The AccuRev repository on **replhost** now has an up-to-date site slice, but the repository doesn’t yet contain detailed data on any depots.

12. List all the depots in the master repository, by executing this command on **replhost**:

```
accurev show -fix depots
```

In the XML-format output, the depots that exist in the master repository, but are not replicated on **replhost**, are listed with this attribute:

```
ReplStatus = "missing"
```



13. For each depot that is to be replicated on **replhost**, execute a **mkreplica** command on **replhost**. For example, if depots named **widget**, **gadget**, and **cust\_support** are to be replicated:

```
accurev mkreplica -p widget
accurev mkreplica -p gadget
accurev mkreplica -p cust_support
```

## Using the Same Host as Both Master Server and Replica Server

In a production environment, it doesn't make sense to have the same machine act as both the master server and a replica server. But it *does* make sense to use a single machine to test new software or work processes. You can use the same machine as both **masthost** and **replhost** in the procedures described in sections *Setting Up the Master Server* and *Setting Up the Replica Server* above. If you do so, keep these points in mind:

- There must be two separate AccuRev installations on the machine.
- Each installation has its own **acserver.cnf** configuration file, located in the AccuRev **bin** directory. The **SITE\_SLICE\_LOC** setting must be different in each **acserver.cnf** file. Similarly, the **DEPOTS\_DEFAULT** setting must be different.
- There is no way to have a depot slice be associated with multiple repositories. A depot slice can be located outside the area specified by the **DEPOTS\_DEFAULT** setting — with **mkdepot -l** on the master server; with **mkreplica -l** on a replica server; or with **chslice** on either kind of server. But don't use these techniques to make multiple instances of the AccuRev Server think that they are managing the same slice location.

## Setting Up a Client Machine to Use a Replica Server

A machine on which the AccuRev client software is installed can use any server — either a replica server or the master server. As always, the **SERVERS** setting in the client configuration file — **acclient.cnf** in the AccuRev **bin** directory — specifies which AccuRev Server process is to be sent client command requests. Examples:

```
SERVERS = replhost:5050           (use replica server)
```

```
SERVERS = masthost:5050          (use master server)
```

You can switch a client back and forth among multiple replica servers (and possibly the master server, too). It's as simple as editing the client's **acclient.cnf** file.

## Using a Replica Server

When your client machine is set up to use a replica server, you can issue all AccuRev commands in the usual way. In general:

- Configuration management commands that *read* data from the repository — such as **files**, **diff**, and **cat** — use the replica repository.

- Configuration management commands that *write* data to the repository — such as **keep**, **promote**, and **merge** — use the master replica. After the master replica has been modified, the local replica is automatically brought up to date. For details, see [Synchronizing a Replica Manually](#) on page 268, which describes how you can bring the local replica up to date when you are *not* writing data to the repository.
- All AccuWork issue management operations are handled by the master server. Thus, replication does not improve AccuWork performance.

## The Update Command

The **update** operation is somewhat of a special case. Currently, an **update** executed on a replica server works as follows:

- A **stat** operation is performed on the replica server, to determine the state of the workspace stream and its backing stream.
- Data files representing new versions of elements are copied from the file storage area in the master repository to your workspace tree.
- Database transactions are copied from the master repository to the replica repository, bringing the replica database completely up to date.
- Storage files corresponding to those database transactions are not copied to the replica repository. See [On-Demand Downloading of a Version's Storage File](#) on page 268.
- The transaction level of the workspace is set to the most recent transaction (or to the transaction specified with **update -t**).

Note: suppose a client machine is using the master server, and you update a workspace using that machine. If you then switch the client machine to using a replica server, you must perform a **replica sync** command before updating the workspace again.

## Creating New Depots

New depots can be created only in the master repository, not in a replica repository. If a client using a replica repository issues a **mkdepot** command, an error occurs:

```
Cannot create a new depot on the replica server
```

After creating a new depot in the master repository, you can include it in a replica repository with this sequence of commands, issued on a client that uses the replica server:

```
accurev replica sync
accurev mkreplica -p <depot-name>
```

## Adding and Removing Depots from a Replica Repository

After you have set up a replica repository, you can use the commands **mkreplica** and **rmreplica** to change which depots are included in the replica repository. These commands are described in the *AccuRev User's Guide (CLI Edition)*.

## Synchronizing a Replica Manually

During the course of development, your local replica repository typically becomes out-of-date with respect to the master repository. This occurs when other users send commands to other replica servers or directly to the master server. In both such cases, new transactions are entered in the master repository, but are not entered in the your local replica repository.

At any time, you can enter this CLI command to bring your local replica repository up to date:

```
accurev replica sync
```

This transfers data from the master repository site slice to the replica repository site slice. It also transfers database transactions from the master repository to the replica repository — but only for the depots that are included in the local replica. It does not transfer the corresponding storage files for **keep** transactions. See *On-Demand Downloading of a Version's Storage File* below.

A **replica sync** command is performed automatically on the local replica after each operation that is initiated by a client of the local replica, and that makes a change to the repository. See *Using a Replica Server* on page 266.

Note: you never need to synchronize directly with other replicas; synchronizing with the master is sufficient to bring your replica up to date.

## On-Demand Downloading of a Version's Storage File

As a performance optimization, AccuRev copies database transactions only — not storage files that hold the contents of **keep** versions — when it transfers data from the master repository to a replica repository. Such transfers take place:

- ... during a **replica sync** command.
- ... during the automatic replica synchronization that follows an operation, invoked by a client using a replica server, that modifies the repository.
- ... during an **update** of a workspace. (**update** copies files to the workspace tree on the client machine, but not to the repository's file storage area on the replica server.)

Storage files can be quite large — particularly images and video clips — and many of them may never need to be accessed by users. For example, suppose a user creates intermediate versions 5,6,7, and 8 of an element, before finally creating and promoting version 9. Chances are no one will ever need to access the contents of versions 5–8 again. (But the master repository does preserve those versions' storage files, just in case!)

The storage file for a version is downloaded from the master repository to the local replica repository when a client using that replica explicitly references that version. Examples:

```
accurev cat -v talon_dvt/12 foo.c
accurev diff -v talon_dvt/12 foo.c
```

Both these commands cause the storage file for version **talon\_dvt/12** of file **foo.c** to be downloaded to the local replica before the command itself is executed.

## Automating Replica Synchronization

If a workgroup is much less active than other workgroups, its local replica repository can “fall behind” the master repository significantly. This can also occur if the workgroup uses the local replica repository mostly as a reference — for frequent read operations, but infrequent write operations. Falling behind in this way does no harm, but it can be bothersome. When some user finally does perform a write operation — keeping a new version of a file, or changing the location of a workspace — the local replica repository automatically “catches up”, which might involve downloading tens or hundreds of transactions.

To prevent the local replica repository from falling too far behind, we recommend that you use operating system tools to perform an **accurev replica sync** command automatically, at regular intervals — say, every 15 minutes. On a Windows machine, use the Scheduled Tasks applet in the Control Panel. On a Unix/Linux host, set up a **cron** job to execute this command.

## Synchronization Security

Note: this section describes a security risk that exists only for organizations using the *AccuRev Replication Server* product. This risk does not apply to organizations that use the standard AccuRev software, without the replication option.

The repository synchronization scheme poses a potential security risk: the **acserver.cnf** server configuration file on an AccuRev server machine can name *any* master server machine in a MASTER\_SERVER setting. And by default, the targeted master server will comply with *any* synchronization request — even an **accurev replica sync** command executed on a completely unrelated client machine.

We strongly recommend using the **server\_admin\_trig** trigger on the master server machine to implement an authentication scheme, so that the master server will send repository data over the wire only to valid requestors. The following Perl code might be added to the sample **server\_admin\_trig** script included in the **examples** subdirectory of the AccuRev distribution:

```
if ($command eq "replica_sync") {
    if ($principal ne "rep01_acadmin" and $principal ne "rep02_acadmin") {
        print TIO "Repository synchronization disallowed:\n";
        print TIO "Authentication by the server_admin_trig script failed.\n";
        close TIO;
        exit(1);
    }
}
```

This code allows users **rep01\_acadmin** and **rep02\_acadmin** to perform repository synchronization, rejecting requests from all other user identities.

Note: a **server\_admin\_trig** script identifies the command as **replica\_sync**, even though the actual CLI command is **replica sync**.

## The replica\_site.xml File

Each replica repository's site slice directory contains an XML-format file, **replica\_site.xml**. This file contains information about the depots that are replicated in that repository. The **mkreplica** and **rmreplica** commands maintain the contents of this file.

# Moving the AccuRev Server and Repository to Another Machine

The AccuRev data repository should be physically located on the machine that runs the AccuRev Server process. (This is firm dictate, but not an absolute restriction — see *READ ME NOW: Assuring the Integrity of the AccuRev Repository* on page 239.) The repository consists of multiple slices: the site slice contains information that pertains to the entire repository, and each depot has its own slice. Each slice contains a database consisting of multiple files.

From time to time, you may want (or need) to have the AccuRev server process run on a different machine. To accomplish this, you must:

- Perform a “full” (client and server) installation of AccuRev on the new machine.
- Move the data repository to the new machine.

If the new machine has a different byte order than the old machine, you must migrate each slice to use the “opposite-endian” data format. This involves swapping the bytes in each machine-level word. The **maintain migrate** command performs this conversion. With no arguments, it migrates the site slice. With an argument, it migrates a depot slice. The results of the migration are stored in a new subdirectory named **swapped** within the site-slice or depot directory.

This procedure is safe: the original slice data is never modified, and there is no harm in running the **maintain migrate** multiple times on a slice.

## Procedure for Moving the Repository

Make sure you perform each of the following steps on the appropriate server machine. We call them:

- The “source” machine — where the AccuRev server is currently running and the data repository is currently located.
- The “destination” machine — the machine to which you want to move the data repository.

Note: the steps below always show Unix pathname separators ( / ). When you’re executing commands on a Windows machine (either source or destination), be sure to use Windows pathname separators ( \ ).

The procedure calls for multiple stops and starts of the AccuRev Server process. For details on how to accomplish this, see *Controlling Server Operation* on page 248.

## On the Destination Machine

1. Perform that a “full” (that is, both client and server) installation of AccuRev on this machine. In the steps below, we’ll refer to the installation directory on the destination machine as *<dest-install-dir>*.

2. Run this command on the destination machine:

```
accurev hostinfo
```

3. Send the output of this command to AccuRev Customer Support, as part of a request for a new license key for the destination machine. When you get the new license key, install it in the **site\_slice** directory, according to the supplied instructions.
4. Store an additional copy of the license key outside the AccuRev repository, for use in Step 14.
5. *Do the following only if you're moving the repository to an opposite-endian machine:*  
Copy the following file to a secure location:

```
<dest-install-dir>/storage/site_slice/datadict.ndb
```

## On the Source Machine

6. Execute the command **accurev show slices** and **accurev show depots**, and save the output for reference in the following steps.
7. Stop the AccuRev Server process. (Reminder: see *Controlling Server Operation* on page 248.)
8. Perform a full backup of the AccuRev repository, as described in *Backing Up the Repository* on page 241.
9. *Do the following only if you're moving the repository to an opposite-endian machine:*

9a) In the **site\_slice** directory, make backup copies in a subdirectory of all the \*.ndb files. For example, back up file **lock.ndb** by copying it to **save\_original/lock.ndb**.

9b) Migrate the site slice, using the **maintain** program located in the AccuRev **bin** directory:

```
maintain migrate
```

This creates a **swapped** subdirectory under the **site\_slice** directory.

9c) Move all \*.ndb database files from the **site\_slice/swapped** directory up to the **site\_slice** directory. This overwrites the original \*.ndb files in the **site\_slice** directory. (These are the files that you backed up in Step 9a.)

9d) For each depot listed in the **show depots** output (see Step 6), make backup copies in a subdirectory of the depot's \*.ndb files. For example, back up file **.../depots/<depot-name>/ancestry.ndb** by copying it to **.../depots/<depot-name>/save\_original/ancestry.ndb**.

9e) For each depot listed in the **show depots** output, migrate the depot's slice:

```
maintain migrate <depot-name>
```

This creates a set of **swapped** subdirectories in the depot slices.

9f) Move all \*.ndb files from the **<depot-name>/swapped** directories up to the parent **<depot-name>** directories. This overwrites the existing \*.ndb files in the **<depot-name>** directories. (These are the files that you backed up in Step 9d.)

10. Copy the entire tree starting at directory **storage** within the AccuRev installation area. Be sure to use a method that preserves file ownership (e.g. **tar -cp**).

## On the Destination Machine

11. Stop the AccuRev Server process.
12. Rename **<dest-install-dir>/storage** (e.g. to **storage.OLD**).
13. “Paste” (unpack, unzip, tar -x) the copy of the directory tree you made in Step 10, so that the pasted data becomes **<dest-install-dir>/storage**.
14. Overwrite file **<dest-install-dir>/storage/site\_slice/keys.txt** with the copy of the license key that you saved in Step 4.
15. *Do the following only if you’re moving the repository to an opposite-endian machine:*
  - 15a) Restore the copy of **<dest-install-dir>/storage/site\_slice/datadict.ndb** that you made in Step 5a.
  - 15b) Reindex the site slice:  

```
maintain reindex
```
16. If the **storage** directory is located at a different pathname on the source and destination machines:
  - 16a) Start the AccuRev Server process.
  - 16b) For each depot, determine the corresponding slice number (see Step 6) and run this command:  

```
accurev chslice -s <slice-number>  
-l <dest-install-dir>/storage/depots/<depot-name>
```
  - 16c) Stop the AccuRev Server process.  
*Perform the following step only if you’re moving the repository to an opposite-endian machine:*
    - 16d) Reindex each depot:  

```
maintain reindex <depot-name>
```
17. Start the AccuRev Server process.





# AccuRev Triggers

A trigger is a “code hook” or callback built into certain AccuRev commands. When a user enters the command, the corresponding trigger “fires”; this causes a user-defined or built-in procedure to be performed just before or after the command executes. Typically, a user-defined procedure is implemented as a script in the Perl scripting language. Sample Perl scripts are available in the **examples** subdirectory of the AccuRev installation directory.

Note: in this chapter, “trigger script” refers to any executable program, written in any language, that is executed when a trigger fires.

AccuRev supports both pre-operation triggers and post-operation triggers. In addition, there are triggers that integrate AccuRev’s configuration management facility with its issue management facility (AccuWork); these triggers have pre- and post-operation components.

Some triggers are set with the **mktrig** command; others are set by placing the script at a special location.

## Pre-Operation Triggers

The following triggers execute a procedure before the user-requested command executes. Each of these triggers has the ability to cancel execution of the user’s command. Some of the triggers fire on the client machine, and others on the server machine. It’s possible for a single command (e.g. **keep**) to cause triggers to fire both on the client and on the server.

### Client-Side Triggers

The following pre-operation triggers fire on the client machine:

- **pre-create-trig**: fires on the client machine prior to execution of an **add** command.
- **pre-keep-trig**: fires on the client machine prior to execution of a **keep** command.
- **pre-promote-trig**: fires on the client machine prior to execution of a **promote** command.

### Server-Side Triggers

The following pre-operation triggers fire on the server machine.

- **server\_admin\_trig**: fires on the server machine prior to execution of certain commands. This is a repository-wide trigger — it fires no matter what depot, if any, the user’s command applies to. The following commands cause **server\_admin\_trig** to fire:

<code>mkdepot</code>	<code>mktrig</code>	<code>mkuser</code>
<code>chdepot</code>	<code>rmtrig</code>	<code>chuser</code>
<code>chslice</code>	<code>lock</code>	<code>chpasswd</code>
<code>mkstream</code>	<code>unlock</code>	<code>lsacl</code>
<code>chstream</code>	<code>defcomp</code>	<code>setacl</code>

mkws	repl_sync	mkgroup
chws	write_schema	addmember
chref		ismember
remove		rmmember
reactivate		

The **defcomp** command is not user-visible; it's used in the implementation of the include/exclude facility CLI commands **incl**, **excl**, and **incldo**. The **repl\_sync** command recognized by the **server\_admin\_trig** trigger corresponds to the CLI command **replica sync**.

Note: prior to Version 3.5, the services now provided by **server\_admin\_trig** were provided by a trigger named **server\_all\_trig**. (In the AccuRev-provided sample trigger script, this includes allowing commands to be executed only by a specified list of AccuRev users.) For backward compatibility, the older trigger is still supported.

- **server\_preop\_trig**: fires on the server machine prior to execution of certain commands. This is a depot-specific trigger — it fires only for commands that operate on the depot(s) where the trigger has been activated. The following commands cause **server\_preop\_trig** to fire:

```
addpromote
keeppurge
```

The **server\_admin\_trig** and **server\_preop\_trig** triggers are designed to be mutually exclusive — for a given command, at most one of these triggers fires. (For many commands, neither of these triggers fires.)

## Post-Operation Triggers

The following triggers execute a procedure after the user-requested command executes successfully. If the user's command fails, the post-operation trigger does not fire. A post-operation trigger always fires on the server machine.

- **server-post-promote-trig**: fires on the server machine subsequent to execution of a **promote** command.
- **server\_dispatch\_post**: fires on the server machine each time a AccuWork issue record is created or modified. This trigger is intended to enable email notification of A sample Perl script is available in the **examples/dispatch** subdirectory of the AccuRev installation directory.

## Trigger for Transaction-Level Integration between Configuration Management and Issue Management

You can achieve tight coordination of your organization's configuration management and issue management capabilities by enabling one or both of the integrations between AccuRev's

configuration management and issue management facilities. The transaction-level integration is enabled by a trigger on a depot-by-depot basis:

```
accurev mktrig -p WidgetDepot pre-promote-trig client_dispatch_promote
```

The “client\_dispatch\_promote” integration routine is built into the AccuRev software — no scripts are required — and includes both pre-operation and post-operation components:

1. On the client machine, a user invokes the AccuRev **promote** command.
2. The pre-operation part of the trigger fires on the client machine, prompting the user to specify one AccuWork issue record. If this part of the trigger fails (e.g. the user specifies a non-existent issue record), the **promote** command itself is cancelled.
3. The **promote** command completes, and is recorded in the AccuRev repository as a transaction.
4. The post-operation part of the trigger fires on the server machine, updating the issue record that the user specified by adding the number of the **promote** transaction to the **affectedFiles** field.

If you use the built-in “client\_dispatch\_promote” integration routine as the **pro-promote-trig** trigger, you must not also set a **server-post-promote-trig** trigger. Doing so would suppress the post-operation component of the “client\_dispatch\_promote” routine. For information on handling this situation and other aspects of customizing the transaction-level integration, see *Implementation and Customization of the Transaction-Level Integration* on page 224 of the *AccuWork Issue Management Manual*.

Note: AccuRev features another integration between configuration management and issue management, which works at the change-package level instead of the transaction level. See *Integrations Between Configuration Management and Issue Management* on page 220 of the *AccuWork Issue Management Manual*.

## Preparing to Use an AccuRev-Provided Trigger Script

Sample trigger scripts are installed with AccuRev, in the **examples** subdirectory. These sample scripts are implemented in the platform-neutral Perl scripting language. Use the following procedure to install and use one of these scripts:

1. **Install Perl.** There are many source on the Web for Perl. We recommend the ActivePerl distribution from <http://www.activestate.com>. This distribution includes a conversion utility, **pl2bat**, which makes a Perl script executable under Windows, by embedding the Perl code in a Windows batch file (**.bat**).

Be sure to install Perl on all appropriate machines. Note that some pre-operation triggers run on the client machine, while others run on the server machine. All post-operation triggers run on the AccuRev server machine.

2. **Get a copy of the sample script.** Copy the sample script from the **examples** subdirectory of the AccuRev installation directory to an AccuRev workspace. Then use the **add** command to place the script under version control.

3. **Prepare the script.** Open the script in a text editor, and customize the script according to the instructions included as comment lines. Before embarking on heavy script customization, be sure to read *The Trigger Parameters File* on page 279.
4. **Enable the Trigger.** Enable the trigger, either with the **mktrig** command or by placing the script in the proper location. See the following section for details.

## Enabling a Trigger

Depending on its type, an AccuRev trigger is enabled in one of these ways:

- Executing an **accurev mktrig** command, specifying the location of the script. AccuRev simply records the location you specify in the repository; it doesn't make a copy of the script. Make sure that no one moves it!
- Placing the executable script file in the location prescribed for that type of trigger.

For details, consult the appropriate subsection below:

### pre-create-trig, pre-keep-trig, pre-promote-trig, server-post-promote-trig

Use the **mktrig** command to enable use of the script in a particular depot. For example:

```
accurev mktrig -p WidgetDepot pre-keep-trig /usr/ac_scripts/addheader
```

The **-p** option isn't necessary if your current directory is in a workspace associated with that depot. When the trigger fires, AccuRev will search for the script at the specified pathname (in the example above, **/usr/ac\_scripts/addheader**).

We strongly suggest specifying an absolute pathname. Otherwise, when the trigger files AccuRev will use the search path of the user (**pre-create-trig**, **pre-keep-trig**, or **pre-promote-trig**) or the search path of the AccuRev Server (**server-post-promote-trig**) to find the specified script file.

### server\_admin\_trig

Place an executable file in subdirectory **triggers** of the **site\_slice** directory:

- Unix: the file must be named **server\_admin\_trig** or **server\_admin\_trig.pl**
- Windows: the file must be named **server\_admin\_trig.bat**

Example:

```
C:\Program Files\AccuRev\storage\site_slice\triggers\server_admin_trig.bat
```

Note: if the directory contains both a **server\_admin\_trig** executable and an old **server\_all\_trig** executable, only the **server\_all\_trig** script is executed.

### server\_preop\_trig

Place an executable file in subdirectory **triggers** of the slice directory of one or more depots (**accurev show slices** displays slice directory locations):

- Unix: the file must be named **server\_preop\_trig** or **server\_preop\_trig.pl**

- Windows: the file must be named **server\_preop\_trig.bat**

Example:

```
/opt/accurev/storage/depots/talon_tests/triggers/server_preop_trig
```

## server\_dispatch\_post

Place an executable file in the AccuRev executables (**bin**) directory on the AccuRev Server machine:

- Unix: the file must be named **server\_dispatch\_post** or **server\_dispatch\_post.pl**
- Windows: the file must be named **server\_dispatch\_post.bat**

Note: for compatibility with previous AccuRev releases, the script can also be named **dispatch\_email**, with the appropriate suffix.

Example:

```
C:\Program Files\AccuRev\bin\server_dispatch_post.bat
```

## Notes on Triggers in Multiple-Platform Environments

Observe these guidelines in a multiple-platform environment, where the trigger script will be accessed from both Windows machines and Unix machines:

- Don't even try to arrange for exactly the same script file to be accessed by all users, on all platforms. Instead, place scripts to be accessed by Unix users in one directory and equivalent scripts to be accessed by Windows users in another directory.
- Make sure the Windows script has a **.bat** suffix (e.g. **check\_for\_comments.bat**), and that the Unix script has no suffix (**check\_for\_comments**).

Note: on a Windows machine, AccuRev searches for a trigger script file named **check\_for\_comments** before it searches for a batch file named **check\_for\_comments.bat**. That's another reason to place Windows and Unix scripts in separate directories.

- Make sure the scripts run correctly on their respective platforms. And remember to revise *all* versions of the script when you revise any one of them!
- In the **mktrig** command, specify the script name without a suffix — e.g.:

```
accurev mktrig -p WidgetDepot pre-keep-trig check_for_comments
```

## The Trigger Parameters File

When a trigger fires and executes a user-supplied script, AccuRev passes two arguments to the script:

- The first argument is the pathname of a flat-text file containing information about the transaction that is about to be performed (or was just completed).

- The second argument is the pathname of an XML-format file containing the same information. (In some cases, detailed below, the XML-format file contains a small amount of additional information that is not contained in the flat-text file.)

Exceptions: only one argument, the pathname of an XML-format file, is passed to a **server\_preop\_trig** script or a **server\_admin\_trig** script.

We use the term trigger parameters file to describe both these files. The flat-text file contains a series of values — usually one value per line — in a prescribed order. The XML-format file contains a set of elements below the top-level **<triggerInput>** element. Each such element contains the information for one parameter: the parameter name is the element tag, the parameter value is the element contents (sometimes encoded as a set of subelements). For example, here are two trigger parameters files generated by the same user command:

Flat-text trigger parameters file	XML-format trigger parameters file
pre-create	<triggerInput>
talon	<hook>pre-create</hook>
talon_dvt_john	<depot>talon</depot>
4	<stream1>talon_dvt_john</stream1>
adding some files	<changePackages></changePackages>
this multi-line	<comment>adding some files
comment has	this multi-line
four lines	comment has
C:/wks/talon/dvt_john	four lines</comment>
john	<topDir>C:/wks/talon/dvt_john</topDir>
/tools/cont.sh	<principal>john</principal>
/tools/end.sh	<elemList>
/tools/start.sh	<elem>/tools/cont.sh</elem>
	<elem>/tools/end.sh</elem>
	<elem>/tools/start.sh</elem>
	</elemList>
	</triggerInput>

The information contained in the trigger parameters file varies among the trigger types, as described in the following sections.

### Format of the “pre-create-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **pre-create-trig** script. This information describes the creation of one or more new elements to a depot (CLI: **add**, GUI: **Add to Depot**).

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it’s also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: <b>pre-create</b>
depot	Name of depot targeted by the command
stream1	Name of the workspace stream in which the new elements are to be created
changePackages	(XML-format parameters file only) The change packages affected by this command; currently defined to be empty for a <b>pre-create-trig</b> trigger
comment	Zero or more comment lines specified by the user (see <i>Encoding of Command Comments</i> on page 288 below)
topDir	Pathname to the top-level directory of the user's workspace tree, as it is listed by the <b>show wspaces</b> command.
principal	AccuRev username of person invoking the command
elemList	One or more files/directories to be added to the depot. For general notes, see <i>Encoding of Element Lists</i> on page 288 below. Notes related to this trigger type appear after this table.

In the flat-text trigger parameters file, the elements to be created are listed, one per line, at the end of the file:

```

/tools/cont.sh
/tools/end.sh
/tools/start.sh
      <-- end-of-file of trigger parameters file

```

There is no need to supply an element count, since an end-of-file condition signals the end of the element list.

In the XML-format trigger parameters file, the element paths are encoded as **<elem>** sub-elements of **<elemList>**:

```

<elemList>
  <elem>/tools/cont.sh</elem>
  <elem>/tools/end.sh</elem>
  <elem>/tools/start.sh</elem>
</elemList>

```

## Overwriting the Trigger Parameters File

A **pre-create-trig** script must overwrite its flat-text parameters file with data that indicates the type of each element to be created. Each line must describe one new element:

```

<element-pathname> <element-type>

```

... where *<element-pathname>* is a pathname from the input “elemList”, and *<element-type>* is a numeric code:



- 1directory
- 2text file
- 3binary file

For example, a command that adds two text files, two binary files, and a directory to the depot would replace its flat-text parameters file with this data:

```
/tools/end.sh 2
/tools/icons 1
/tools/icons/end.png 3
/tools/icons/start.png 3
/tools/start.sh 2
```

Note: there is currently no provision for the script to overwrite the XML-format trigger parameters file. The data to be passed to the AccuRev Server must be in flat-text format.

## Format of the “pre-keep-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **pre-keep-trig** script. This information describes the creation of a new versions of one or more existing elements in a depot (CLI: **keep**, GUI: **Keep**).

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it’s also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: <b>pre-keep</b>
depot	Name of depot targeted by the command
stream1	Name of the workspace stream in which the new versions are to be created
changePackages	(XML-format parameters file only) The change packages affected by this command; currently defined to be empty for a <b>pre-keep-trig</b> trigger
comment	Zero or more comment lines specified by the user (see <i>Encoding of Command Comments</i> on page 288 below)
topDir	Pathname to the top-level directory of the user’s workspace tree, as it is listed by the <b>show wspaces</b> command.
principal	AccuRev username of person invoking the command
elemList	A specification for each new element version to be created. For general notes, see <i>Encoding of Element Lists</i> on page 288 below. Notes related to this trigger type appear after this table.

In the flat-text trigger parameters file, the versions to be created are listed, one per line, at the end of the file. Each line contains three specifications:

```
<element-pathname> <version-ID> <element-type>
```

There is no need to supply an element count, since an end-of-file condition signals the end of the element list. For example:

```
/tools/icons/end.png talon_dvt_john/5 3
/tools/icons/end.sh talon_dvt_john/9 2
/tools/icons/start.png talon_dvt_john/2 3
/tools/icons/start.sh talon_dvt_john/13 2
```

In the XML-format trigger parameters file, each version to be created is encoded as an **<elem>** sub-element of **<elemList>**. The element’s attributes specify the version-ID (**stream** and **version** attributes) and the element-type (**elemType** attribute). The element pathname is encoded as the contents of **<elem>**.

The following example contains the same data as the flat-text example above:

```
<elemList>
  <elem
    stream="talon_dvt_john"
    version="5"
    elemType="3"/>/tools/icons/end.png</elem>
  <elem
    stream="talon_dvt_john"
    version="9"
    elemType="2"/>/tools/icons/end.sh</elem>
  <elem
    stream="talon_dvt_john"
    version="2"
    elemType="3"/>/tools/icons/start.png</elem>
  <elem
    stream="talon_dvt_john"
    version="13"
    elemType="2"/>/tools/icons/start.sh</elem>
</elemList>
```

In either format, the element-type value can be either **2** (text file) or **3** (binary file). Note that different versions of an element can have different types.

## Format of the “pre-promote-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **pre-promote-trig** script. This information describes the creation of a new versions of one or more existing elements in a depot (CLI: **promote**, GUI: **Promote**).

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it’s also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: <b>pre-promote</b>

Parameter	Description
depot	Name of depot targeted by the command
stream1	Name of the workspace or stream that the versions are to be promoted from
changePackages	(XML-format parameters file only) The change packages affected by this command; currently defined to be empty for a <b>pre-promote-trig</b> trigger
comment	Zero or more comment lines specified by the user (see <i>Encoding of Command Comments</i> on page 288 below)
topDir	Pathname to the top-level directory of the user's workspace tree, as it is listed by the <b>show wspaces</b> command.
principal	AccuRev username of person invoking the command
elemList	One or more files/directories to be promoted. For general notes, see <i>Encoding of Element Lists</i> on page 288 below. Notes related to this trigger type appear after this table.

In the flat-text trigger parameters file, the elements to be created are listed, one per line, at the end of the file:

```
/tools/cont.sh
/tools/end.sh
/tools/start.sh
<-- end-of-file of trigger parameters file
```

There is no need to supply an element count, since an end-of-file condition signals the end of the element list.

In the XML-format trigger parameters file, the element paths are encoded as **<elem>** sub-elements of **<elemList>**:

```
<elemList>
  <elem>/tools/cont.sh</elem>
  <elem>/tools/end.sh</elem>
  <elem>/tools/start.sh</elem>
</elemList>
```

## Overwriting the Trigger Parameters File

A **pre-promote-trig** script can work in tandem with a **server-post-promote-trig** script, providing customized “before and after” processing around the execution of **Promote** commands.

A **pre-promote-trig** script can overwrite its flat-text triggers parameters file, in order to communicate with a **server-post-promote-trig** script: the *first line* of the overwritten parameters file becomes the value of the *<fromClientPromote>* parameter passed to the **server-post-promote-trig** script.

Note: there is currently no provision for a **pre-promote-trig** script to pass data to a **server-post-promote-trig** script by overwriting the XML-format trigger parameters file.

## Format of the “server-post-promote-trig” Trigger Parameters File

The following table presents the information in the trigger parameters file sent to a **server-post-promote-trig** script. This information is generated by AccuRev, and describes the **Promote** command that has just executed. The file can also contain a user-specified text string, generated by a **pre-promote-trig** script, to the **server-post-promote-trig** script.

The order of the parameters in this table is the order in which they appear in the flat-text trigger parameters file. (Less importantly, it’s also the order in which they appear in the XML-format trigger parameters file.)

Parameter	Description
hook	Type of trigger: <b>server-post-promote</b>
depot	Name of depot targeted by the command
stream1	Name of the stream that the versions were promoted to
fromClientPromote	A single text line (including the line-terminator): the first line of the trigger parameters file passed to the <b>pre-promote-trig</b> script. See the notes after this table.
changePackages	A set of <changePackageID> subelements, specifying the change packages (that is, issue records) specified in the user’s command.
transNum	The transaction number of the Promote transaction that just completed.
transTime	The time of the Promote transaction that just completed.
comment	Zero or more comment lines specified by the user in the Promote command (see <i>Encoding of Command Comments</i> on page 288 below)
topDir	Pathname to the top-level directory of the user’s workspace tree, as it is listed by the <b>show wspaces</b> command.
principal	AccuRev username of person invoking the command
elemList	A specification for each version that was promoted. For general notes, see <i>Encoding of Element Lists</i> on page 288 below. Notes related to this trigger type appear after this table.

## Format of the “server\_preop\_trig” Trigger Parameters File

The parameters file passed to a **server\_preop\_trig** script is in XML format:

```
<triggerInput>
  <hook> ... </hook>
  <command> ... </command>
  <principal> ... </principal>
  <ip> ... </ip>
```

...  
</triggerInput>

The set of subelements under the **<triggerInput>** element depends on the user's command. The following table provides a summary. For full details, see the sample **server\_preop\_trig** script in the **examples** directory in the AccuRev installation area.

Parameter	Description
hook	Type of trigger: <b>server_preop_trig</b>
command	The user command: <b>add</b> , <b>keep</b> , <b>promote</b> , or <b>purge</b>
principal	AccuRev username of person invoking the command
ip	The IP address of the client machine
stream1	The user's workspace stream
stream2	The workspace's parent (backing) stream
depot	Name of depot targeted by the command
fromClientPromote	(Promote command only) The number of the AccuWork issue record entered by the user, when prompted by the transaction-based integration or the change-package-based integration.
changePackagePromote	(Promote command only) A set of <changePackageID> subelements, specifying the change packages (that is, issue records) specified in the user's command.  These forms of the promote command generate a <changePackagePromote> element: <ul style="list-style-type: none"><li>• promote -I &lt;issue-number&gt;</li><li>• promote (user prompted by issue-management integration to specify an issue record)</li><li>• promote -Fx (user specifies a set of change packages with an XML file)</li></ul>
comment	Comment string specified by the user. If the comment spans multiple lines, line-terminators are embedded in the string, but the final line does not have a line-terminator.
elemList	A set of <elem> subelements, each specifying one element processed by the user's command.

## Format of the “server\_admin\_trig” Trigger Parameters File

The parameters file passed to a **server\_admin\_trig** script is in XML format: The set of subelements under the **<triggerInput>** element depends on the user's command. The following table provides a summary. For full details, see the sample **server\_preop\_trig** script in the **examples** directory in the AccuRev installation area.

Parameter	Description
hook	Type of trigger: <b>server_admin_trig</b>
command	The user command
principal	AccuRev username of person invoking the command
user	(chuser, chpasswd) AccuRev username being modified
newName	(chuser) new AccuRev username
newKind	(chuser) new user kind ( <b>dispatch</b> or <b>cm</b> )
ip	The IP address of the client machine
stream1	The stream targeted by the user command
stream2	The parent (backing) stream of <b>stream1</b>
stream3	(chws, chstream) The new name of the workspace or stream
objectName	(remove, reactivate) Name of object targeted by the command
objectType	(remove, reactivate) Type of object targeted by the command: <b>1</b> =reference tree; <b>2</b> =workspace; <b>3</b> =stream; <b>5</b> =user; <b>6</b> =group

### Format of the “server\_dispatch\_post” Trigger Parameters File

The parameters file passed to a **server\_dispatch\_post** script is in flat-text format. The order of the parameters in the table below is the order in which they appear in the file.

Parameter	Description
hook	Type of trigger: <b>server-post-dispatch</b>
project	Name of depot in which the AccuWork issues database resides
stream	blank line
from_client_promote	Two SPACE-separated fields: <ul style="list-style-type: none"> <li>The number of the transaction that created the previous version of this issue record. (The number <b>0</b> indicates that this is a newly created issue record.)</li> <li>The issue number</li> </ul>
transaction_num	The number of the transaction that created this new version of the issue record
transaction_time	The time at which <b>transaction_num</b> was created
comment_lines	blank line
author	The AccuRev principal-name of the user who saved the issue record.

## Encoding of Element Lists

In both kinds of trigger parameters files, each element is listed by its path relative to the depot's top-level directory:

```
/tools/cont.sh
```

The path begins with a slash in order to simplify constructing the element's full pathname on the client machine: just append the given element pathname to the **topDir** pathname (the top-level directory of the user's workspace tree).

In the flat-text trigger parameters file, the elements (or elements-to-be) to be processed by the user command are listed, one per line, at the end of the file:

```
/tools/cont.sh
/tools/end.sh
/tools/start.sh
<-- end-of-file of trigger parameters file
```

Unlike the set of comment lines, there is no need to supply an element count; an end-of-file condition signals the end of the element list.

In the XML-format trigger parameters file, the element paths are encoded as **<elem>** sub-elements of the **<elemList>** element:

```
<elemList>
  <elem>/tools/cont.sh</elem>
  <elem>/tools/end.sh</elem>
  <elem>/tools/start.sh</elem>
</elemList>
```

## Encoding of Command Comments

In the flat-text trigger parameters file, the user's comment is indicated by a line-count (0 or greater), followed by the lines of the comment, if any:

```
4                                <-- number of comment lines to follow
adding some files
this multi-line
comment has
four lines
```

In the XML-format trigger parameters file, the user's comment is encoded as the contents of the **<comment>** element: a single string. For a multi-line comment, this string has line-terminators embedded:

```
<comment>adding some files  <-- embedded line-terminator
this multi-line            <-- embedded line-terminator
comment has                <-- embedded line-terminator
four lines</comment>
```

Note that the final line-terminator is automatically stripped from all comment strings.

The sample set of trigger scripts includes a Perl script for each kind of trigger. The script's comments include a detailed description of the layout of the parameters file for that kind of trigger.

## Trigger Script Contents

A trigger script can send email, start a build, update a Web site, or perform many other tasks. In particular, you can run AccuRev commands to get more information. One common use of the **server-post-promote-trig** trigger is to run the **hist** command using the transaction number of the promotion, generating the list of promoted elements for inclusion in an email notification.

## Trigger Script Exit Status

The exit status (return value) of a **pre-create-trig**, **pre-keep-trig**, or **pre-promote-trig** script is important:

- A zero exit status indicates success: the AccuRev command (**add**, **keep**, or **promote**) is allowed to proceed.
- A non-zero exit status indicates failure: the AccuRev command is canceled and the depot remains unchanged.

## File Handling by Trigger Scripts

A trigger script can overwrite its parameters file (after reading it, presumably). This provides a way for the script to communicate with the AccuRev command or with a “downstream” script:

- The parameters file for a **pre-keep-trig** script ends with a series of lines, one per element to be kept:

*<pathname-of-element> <version-ID> <element-type>*

*<pathname-of-element>* is not a full file system pathname, but starts at the workspace's top-level directory (which is included earlier in the parameters file). *<version-ID>* is the new version to be created for that element. *<element-type>* is the numeric code 1, 2, or 3, as described above. Note that different versions of an element can have different types.

See sample trigger script **addheader.pl** in the **examples** subdirectory of the AccuRev installation directory.

- The parameters file for a **pre-promote-trig** script ends with a series of lines, one per element to be promoted:

*<pathname-of-element>*

*<pathname-of-element>* is not a full fleshiest pathname, but starts at the workspace's top-level directory (which is included earlier in the parameters file).

A **pre-promote-trig** script can overwrite its parameters file, in order to communicate with a **server-post-promote-trig** script: the *first line* of the overwritten parameters file becomes the value of the **from\_client\_promote** parameter in the **server-post-promote-trig** script.



See sample trigger script **client\_dispatch\_promote\_custom.pl** in the **examples/dispatch** subdirectory of the AccuRev installation directory, along with **server\_post\_promote.pl** in the **examples** subdirectory.

A trigger script can also send data to STDOUT and STDERR. If the command for which the trigger fired was executed in the AccuRev CLI, this data appears in the user's command window. If a GUI command caused the trigger to fire, the script's exit status determines whether the user sees the STDOUT/STDERR data: in the "failure" case (non-zero exit status), the data is displayed in an error-message box; in the "success" (zero exit status) case, the data is discarded.

## Trigger Script Execution and User Identities

When a trigger script executes on a client machine, it runs under the identity of the AccuRev user who entered the command. Since the user himself is registered (i.e. has a principal-name) in the AccuRev user registry, there won't be any authentication problems if the trigger script runs AccuRev commands that access the repository.

When a trigger script executes on the server machine, it runs under the user identity of the AccuRev Server itself. We recommend that the server run as user **acserver** or **System** (see *User Identity of the Server Process* on page 245), a user that should not be in AccuRev's user registry. If the trigger script runs AccuRev commands, it needs to have an AccuRev user identity. To accomplish this, set variable ACCUREV\_PRINCIPAL to an appropriate principal-name (AccuRev username) in the environment of the **acserver** or **System** account. Alternatively, set this environment variable in the script itself. For example:

```
$ENV{ACCUREV_PRINCIPAL} = "jjp";
system("accurev hist ...");
```

## 'Administrative Users' in Trigger Scripts

The sample Perl trigger scripts supplied by AccuRev provide a very simple implementation of the "administrative user" concept: a user is permitted to perform certain operations only if his username is recorded in the **administrator** hash defined in the script:

```
$administrator{"derek"} = 1;
$administrator{"allison"} = 1;
...
if ( ! defined($administrator{$principal}) ) {
    print TIO "Execution of '$command' command disallowed:\n";
    ...
}
```

## The Trigger Log File

When a trigger script runs on the AccuRev server machine — for a **server-post-promote-trig**, **server\_preop\_trig**, or **server\_admin\_trig** trigger — an invocation line is written to file **trigger.log** in the **logs** subdirectory of the repository's **site\_slice** directory:

```
##### [2004/06/28 20:50:42] running: 'C:\Program Files\AccuRev\bin\pst_pro.bat' ...
```

If the script produces console output (STDOUT and/or STDERR), this output is also sent to the **trigger.log** file.

As with other server log files, the **trigger.log** file is “rotated” periodically, to keep active logs from growing too large.



# The ‘maintain’ Utility

This document describes AccuRev’s **maintain** utility, an administrative tool for occasional use under the guidance of an AccuRev, Inc. Product Support representative. The **maintain** utility is a non-interactive command-line tool, with a simple command structure. For example:

```
maintain reindex <depot-name>
```

The **maintain** program is located in the AccuRev **bin** directory. If the command-line client program, **accurev**, is on your search path, then so is **maintain**.

Each of the **maintain** commands is described in the next section. Following that are sections providing a more discussions of several of the commands.

## ‘maintain’ Command Reference

### backup

Along with **reindex** and **restore**, provides a facility for backing up and restoring the AccuRev repository. See *Backup/Restore of the AccuRev Repository* on page 295.

### chpasswd

```
maintain chpasswd <user> <new-password>
```

Changes the password stored in the AccuRev repository for an existing principal-name (named AccuRev user). To remove a user’s password, use two consecutive double-quote characters as the *<new-password>* parameter:

```
maintain chpasswd derek ""
```

For consistency, the user should invoke the **accurev setlocalpasswd** command to change his “local password” on each client machine that he uses. A user’s local password is stored in file **authn** in subdirectory **.accurev** of the user’s home directory.

### chuser

```
maintain chuser <user-ID> <new-username>
```

Changes the principal-name (AccuRev username) of an existing user. You specify the user by the unique numeric user-ID, which is immutable. This command is similar to the **accurev chuser** command.

### dbcheck

```
maintain dbcheck
```

Checks the consistency of the streams database (**streams.ndb**) in the **site\_slice** directory, and checks the consistency of all the database (**.ndb**) files in all depots.

Note: be sure to stop the AccuRev Server process before running this command.

## export

```
maintain export
maintain export <depot-name>
```

Produces an ASCII dump of the repository's site slice directory, or an ASCII dump of the specified depot.

## loc128

```
maintain loc128 <depot-name>
```

Changes the maximum length of a pathname segment (the simple name of a file or directory) to 128 characters, for the specified depot.

Use this command to “upgrade” depots created with earlier versions of AccuRev; in later versions, all depots are created with the 128-character maximum for pathname segments.

## migrate

```
maintain migrate
maintain migrate <depot-name>
```

Converts the specified depot from little-endian to big-endian format, or vice-versa. If you omit the *<depot-name>* argument, it converts the **site\_slice** directory. In all cases, the conversion is non-destructive: the data structure itself is not modified or deleted; the converted data is written to a subdirectory named **swapped** within the **site\_slice** directory or depot directory. Converting the “endian-ness” consists of reversing the order of the 8-bit bytes in each machine-level word. For a step-by-step conversion procedure, see *Moving the AccuRev Server and Repository to Another Machine* on page 271 of the *AccuRev Administrator's Guide*.

## reindex, restore

Along with **backup**, provides a facility for backing up and restoring the AccuRev repository. See *Backup/Restore of the AccuRev Repository* on page 295.

## rmdepot

```
maintain rmdepot <depot-name>
```

Removes a depot from the AccuRev repository. All streams, snapshots, and workspace streams are also removed from the repository. (Workspace trees are *not* removed.) For details, see *Removing a Depot from the AccuRev Repository* on page 296.

## timeshift

```
maintain timeshift <depot-name> <earlier-trans-#> <later-trans-#> <seconds>
```

Adds a number of seconds to the timestamps of a range of transactions, for the specified depot. The *<seconds>* parameter must be an integer; use a negative integer to shift timestamps backward. All transactions in the specified transaction range (inclusive) are time shifted.

This command refuses to make any change if any *shifted* transaction would move past any *unshifted* transaction. That is, the command refuses to change the order of transactions in the repository.

## vercheck

```
maintain vercheck [ -c | -q ] [ -e <eid> ] <depot-name>
```

Checks the storage containers in the specified depot's **data** directory tree, to verify that a storage container file (**.sto**) exists for each file version recorded in the depot database. It also reports occurrences of “crc mismatch” problems: the actual checksum (CRC) of a **.sto** file does not match the checksum recorded in the corresponding rapid recovery file (**.rrf**).

In addition, you can correct “crc mismatch” problems, using these options:

- **-q** option: for each file with a “crc mismatch”, (step 1) compute the checksum of the **.sto** file, and (step 2) replace the “c:” value in the **.rrf** file with this newly computed value.
- **-c** option: like **-q**, start with this step for each file: (step 0) change the **.sto** file by removing all its CR characters — that is, all bytes with the value **0x0D**.

You can restrict the processing to versions of a particular element with the **-e** option.

**vercheck** fixes all the versions of the element that have a “crc mismatch” problem, leaving other versions as is.

## Backup/Restore of the AccuRev Repository

An **accurev** command (**backup**) and two **maintain** commands (**restore** and **reindex**) are involved in the scheme for backing up and restoring the AccuRev repository with a minimum of disruption to development activities.

The command **accurev backup mark** declares a “checkpoint” of the entire AccuRev repository, by:

- Copying all the database files (**.ndb**) and the index file **stream.ndx** from the **site\_slice** directory to a subdirectory named **backup**.
- Recording the current state of each depot's database files in file **valid\_sizes\_backup** in the depot directory.

This is an **accurev** command — it can be executed while the AccuRev Server process is running and development activities are ongoing. The **backup mark** command does not actually copy any files to a backup medium; it just marks a consistent state of the repository files. After executing this command, you can back up the repository files, in any order, and on any schedule, while users continue to use AccuRev. For more information, see *Backing Up the Repository* on page 241.

At any time after you've executed an **accurev backup mark** command and copied the repository files to a backup medium, you can restore the repository to its state at the time the **backup** command was executed. This is an offline procedure — the AccuRev Server must be stopped when you run it. The procedure is documented in section *Restoring the Repository* on page 242. The **maintain** commands involved are:

## restore

```
maintain restore <depot-name>
```

Rolls back all the database files (**.ndb**) in the specified depot to their state at the time of the **accurev backup mark** command. This is a “logical truncate” operation — a file-system truncate is *not* performed on the database files.

## reindex

```
maintain reindex [ <depot-name> ]
```

Reads various database files (**.ndb**) and recreates the corresponding index files (**.ndx**). This operation is performed on the database files of the specified depot; if you omit the *<depot-name>* argument, the operation is performed on the repository-wide database files in the **site\_slice** directory

## Removing a Depot from the AccuRev Repository

This section describes a procedure for removing a depot completely from the AccuRev repository. Removing a depot:

- Deletes every version of every file and directory in the depot.
- Deletes the entire history of the depot — all transactions involving the depot and its elements.

Removing a depot does not affect any of the workspaces or reference trees that contain copies of the depot’s elements.

## Before You Begin

We strongly recommend that you preserve a backup copy of the AccuRev data repository before deleting any depots. See *Backing Up the Repository* on page 241. Much of a depot’s data is stored in its slice of the repository. Use the command **accurev show slices** to determine the pathname of a depot’s slice; you’ll need it in Step 3 below.

## Depot Removal Procedure

The following procedure must be performed on the machine where the AccuRev repository resides.

1. Stop the AccuRev Server process:
  - Unix: use the **acservctl** utility, located in the AccuRev **bin** directory:

```
acservctl stop
```
  - Windows: use the **Services** applet, or enter the command **net stop accurev** in a Command Prompt window.
2. Execute the **maintain** program’s remove-depot command. This utility program is located in the AccuRev **bin** directory.

```
maintain rmdepot <depot-name>
```

For safety, the **rmdepot** command goes through two confirmation steps, including having you retype the depot name. The command displays some messages, ending with:

```
Site reindex complete.
```

The **rmdepot** command completely removes the depot's records from the repository database in the **site\_slice** directory. (There's one exception: the depot's name remains in the database. See Step 5.)

3. Remove the depot directory subtree (slice) from the AccuRev **storage** directory tree. Be careful not to remove any other depot's directory subtree! If you're not sure where the depot's slice is located, use the command **accurev show -fi slices** to determine the pathname. (This command requires the AccuRev Server to be running.)

For example, if the depot is named **widget** and AccuRev is installed on a Unix machine at **/opt/accurev**, use this command:

```
rm -r /opt/accurev/storage/depots/widget
```

4. Restart the AccuRev Server process:
  - Unix: use the **acserverctl** utility, located in the AccuRev **bin** directory:

```
acserverctl start
```
  - Windows: use the **Services** applet, or enter the command **net start accurev** in a Command Prompt window.
5. At this point, the depot's name remains in the AccuRev repository database. It won't appear in an **accurev show depots** command listing; but you *can* make it appear in a GUI listing of the repository's depots. If you wish to reuse the removed depot's name, perform a rename-depot command. For example:

```
> accurev chdepot -p widget deleted_widget
Unknown stream or ver spec: widget
Unknown stream or ver spec: widget
Changed name of depot widget to deleted_widget .
```

At this point, you can create a new depot named **widget**, which will be completely unrelated to the depot you removed.

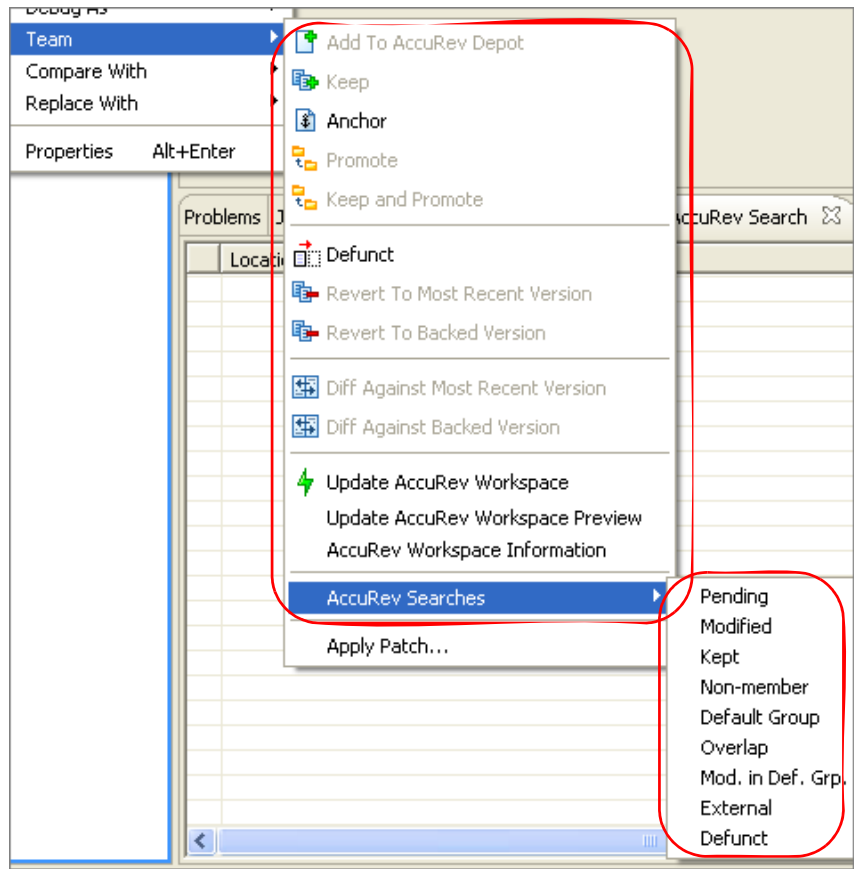




# Using AccuRev with Eclipse

This note describes the integration of AccuRev with the **Eclipse Platform** IDE, implemented with a standard Eclipse “plug-in”. The AccuRev plug-in enables users of the IDE to access AccuRev version-control facilities using the IDE’s own **Team** menu.

As of AccuRev Version 4.0, the AccuRev plug-in has been validated with Eclipse Platform, Version 2.1.2 (using version 1.3.1 of the Java 2 Runtime Environment), and with Eclipse Platform, Version 3.1.0 (using version 1.4.2 of the Java 2 Runtime Environment). This document describes the version of the AccuRev plug-in for use with Eclipse Platform, Version 3.x.



Check file **AccuRevTeamPlugIn-ReadMe.txt** (located in the AccuRev plug-in subdirectory of the Eclipse **plugins** directory) for updated information on platform support.

## Installing the AccuRev Plug-in

The AccuRev plug-in is installed in a subdirectory in the Eclipse **plugins** directory, named **com.accurev.eclipse\_3.8.0**. (The final digits of the directory name may differ; we refer to it as **com.accurev.eclipse\_x.x.x** below.)

Use the following procedure to install this directory in the Eclipse plugins area:

1. Locate the **com.accurev.eclipse\_x.x.x.zip** file. For example, a copy of this file is placed in the AccuRev **bin** directory during AccuRev installation. You may be able to retrieve a newer version from the AccuRev Web site, from your system administrator, etc.

2. Using a tool that can process ZIP files, unpack **com.accurev.eclipse\_x.x.x.zip** to the Eclipse plugins directory. Directory **com.accurev.eclipse\_x.x.x** must be a sibling of all the other Eclipse plugins. The directory hierarchy should look like this:

```
Eclipse-installation-dir/  
  plugins/  
    com.accurev.eclipse_x.x.x/  
      eclipse.jar  
      plugin.xml  
      icons/  
        accli16.gif  
        anchor.gif  
        ...
```

The AccuRev plug-in will be loaded and enabled automatically when you restart Eclipse. To verify successful installation:

1. Start the Eclipse IDE.
2. Select **Help > About Eclipse** from the Eclipse main menu.
3. Click **Plug-in Details**.
4. Verify that an **AccuRev Team Provider** entry exists in the plug-in listing.

## If You're Upgrading ...

If another version of the AccuRev plug-in was previously installed on your machine, use the following command to clear Eclipse's plug-in cache of information related to that other version:

```
eclipse -clean <Eclipse-workspace-path>
```

Start Eclipse in this way for each Eclipse workspace that contains AccuRev-controlled files.

At this point, all existing projects that used AccuRev for version control should work with the newly installed plug-in. In some cases, however, you may find that the AccuRev actions are not enabled under the **Team** context menu. In such cases:

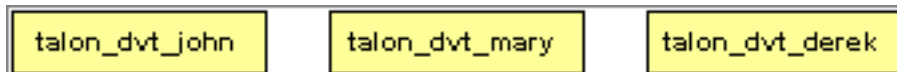
1. Delete the project from the Eclipse workspace, making sure *not* to delete the project's contents.
2. Import the existing project into the Eclipse workspace, using the **File > Import** command.
3. Invoke the **Team > Share Project** command on the project, and specify **AccuRev** as the repository type.

## The AccuRev Usage Model

AccuRev's flexibility makes it easy to use for a variety of development scenarios. But like every software system, AccuRev has usage models that were foremost in the minds of its architects. This section describes the most common usage model.

AccuRev is a software configuration management (SCM) system, designed for use by a team of people (users) who are developing a set of files. This set of files might contain source code in any programming language, images, technical and marketing documents, audio/video tracks, etc. The files — and the directories in which the files reside — are said to be “version-controlled” or “under source control”.

For maximum productivity, the team’s users must be able to work independently of each other — sometimes for just a few hours or days, other times for many weeks. Accordingly, each user has his own private copy of all the version-controlled files. The private copies are stored on the user’s own machine (or perhaps in the user’s private area on a public machine), in a directory tree called a workspace. We can picture the independent workspaces for a three-user team as follows:



Note: an AccuRev workspace is different from an Eclipse workspace — be careful to distinguish them. Any number of AccuRev workspaces can be used with a single Eclipse workspace. The AccuRev plug-in is also compatible with the use of multiple Eclipse workspaces.

This set of users’ workspaces uses the convention of having like names, suffixed with the individual usernames. AccuRev enforces this username-suffix convention. **talon\_dvt** might mean “development work on the Talon product”; **john**, **mary**, and **derek** would be the users’ operating system login names.

From AccuRev’s perspective, development work in this set of workspaces is a continual back-and-forth between “getting in sync” and “getting out of sync”:

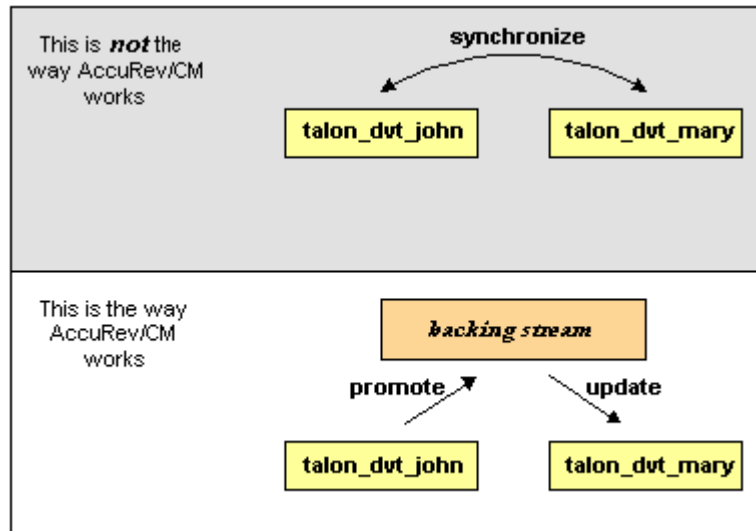
- Initially, the workspaces are completely synchronized: they all have copies of the same set of version-controlled files.
- The workspaces lose synchronization as each user makes changes to some of the files.
- Periodically, users share their changes with each other. When **john** incorporates some or all of **mary**’s changes into his workspace, their two workspaces become more closely (perhaps completely) synchronized.

You might assume that the workspace synchronization process involves the direct transfer of data from one workspace to another. But this is not the way AccuRev organizes the work environment. Instead of transferring data directly between private areas (that is, between users' workspaces), AccuRev organizes the data transfer into two steps:

1. One user makes his changes public — available to all the other members of his team. This step is called promotion.
2. Whenever they wish, other team members incorporate the public changes into their own workspaces. This step is called updating.

The first step involves a public data area, called a stream. AccuRev has several kinds of streams; the kind that we're discussing here is called a backing stream. The data in this public stream “is in back of” or “provides a backstop for” all the private workspaces of the team members. (The terms “parent stream” and “basis stream” are equivalent to “backing stream”.)

AccuRev also allows you to save any number of intermediate versions of a file in your workspace, before making your changes public. Such “private” versions of a file are created by the keep operation.



## Establishing Your Identity

All AccuRev actions must be executed by a user who is listed in the AccuRev user registry. By default, the plug-in uses your operating-system username as your AccuRev username. To have it use a different AccuRev username, set environment variable `ACCUREV_PRINCIPAL` before starting the Eclipse IDE. Here's a Bash Shell example:

```
export ACCUREV_PRINCIPAL=derekp
```

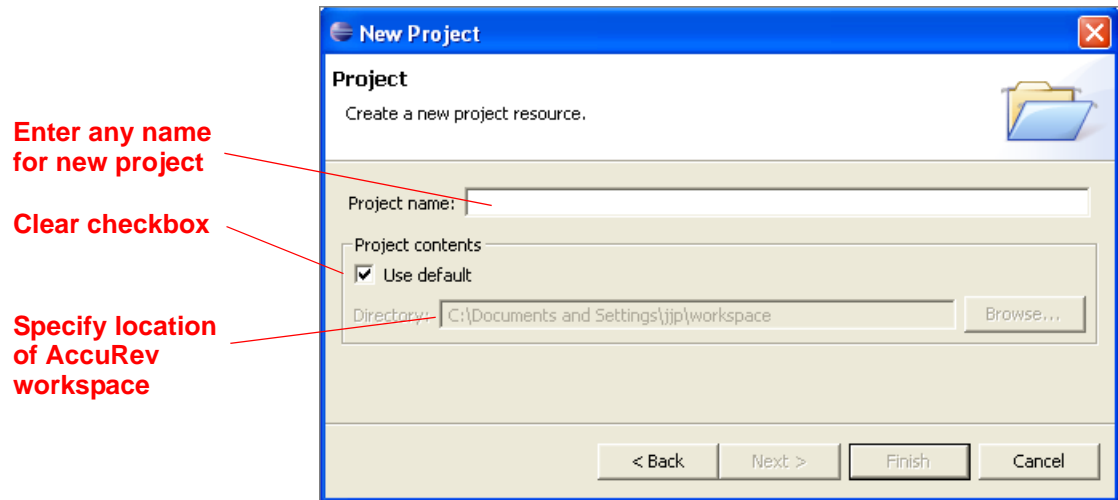
## Creating an Eclipse Project for Your AccuRev Data

The various actions defined by the AccuRev plug-in move data between the central source-code repository (called a depot) and your personal work area (called a workspace in AccuRev, and a project in Eclipse). The AccuRev workspace must be created with the AccuRev GUI or CLI; there is no plug-in action that creates an AccuRev workspace.

To work with the files in a particular AccuRev workspace, you must create an Eclipse project for it. Here's an example:

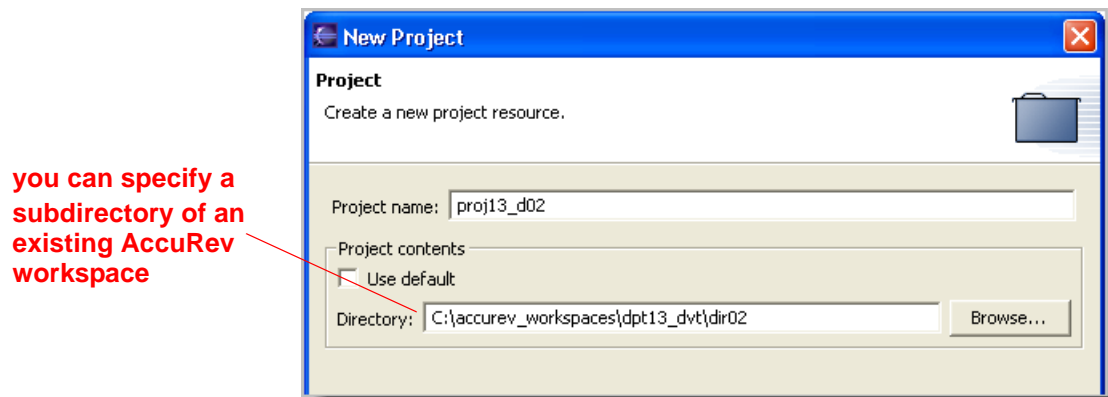
1. From the Eclipse main menu, select **File > New > Project**.

2. Select the **Simple Project** wizard.
3. Enter a project name, then specify the location of an existing AccuRev workspace: clear the **Use default** checkbox, and enter a pathname in the **Directory** field:

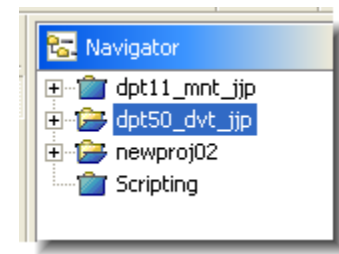


Note this flexibility:

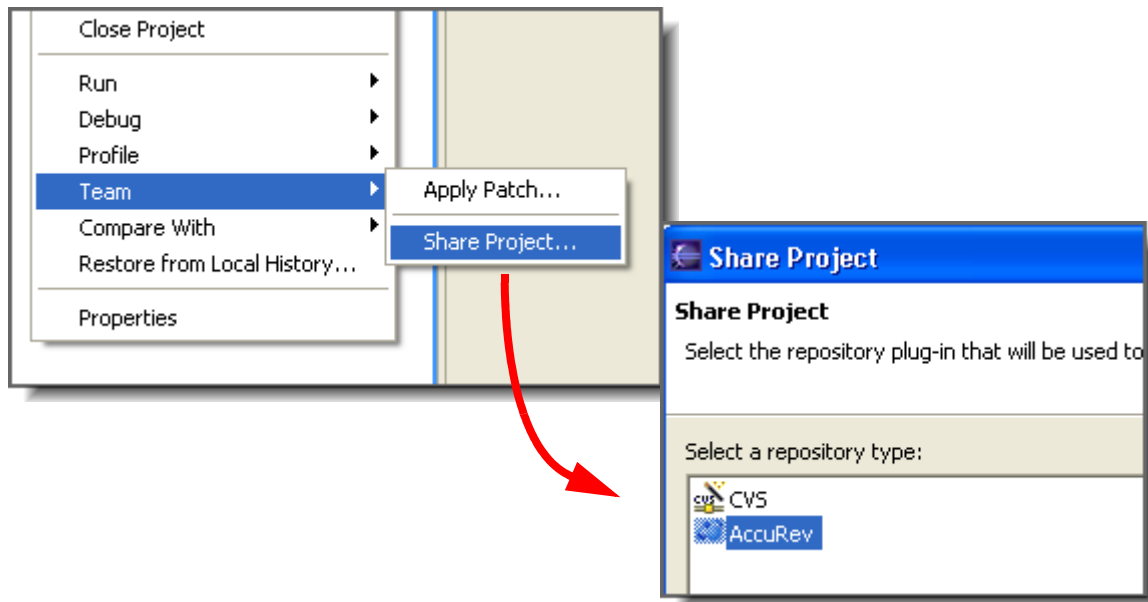
- To create an Eclipse project that contains an *entire* AccuRev workspace, enter the pathname of the workspace's root directory. (This pathname is listed by the AccuRev CLI command **accurev show wspaces**.)
- To create an Eclipse project that contains just a *subtree* of an AccuRev workspace, enter the appropriate pathname *below* the workspace's root directory.



4. Click **Finish** to complete the creation of the project. The new Eclipse project appears in the Navigator view, along with the other projects in the same Eclipse workspace.



5. In the Navigator view, right-click the new project, and select the **Team> Share Project** command from the context menu. Then select **AccuRev** as the provider.



AccuRev actions now appear on the **Team** context menu for the files in this project.

Note: this procedure does not copy any data into your Eclipse **workspace** folder. The data remains in its original location in the AccuRev workspace, and simply becomes accessible through the Eclipse project.

## Advanced “Pick and Choose” Techniques

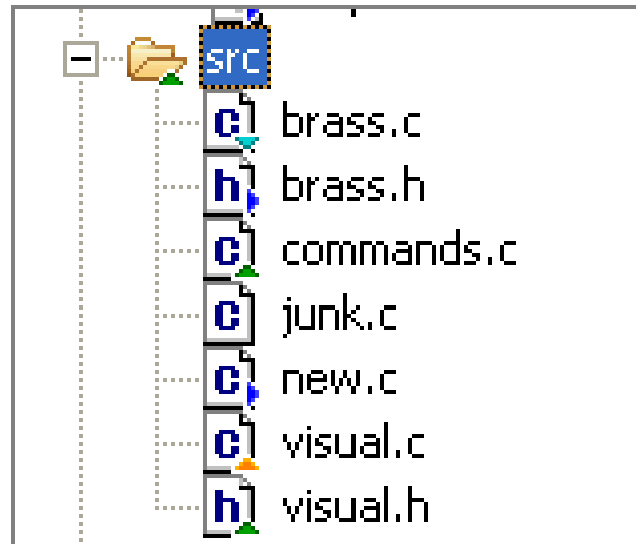
If you want to work with several distinct subtrees of an existing AccuRev workspace, you must create a separate Eclipse project for each subtree. When performing AccuRev actions that search through all the files in a project, you can choose to include all such “sibling” projects in a single search. See [Searches on Parts of a Workspace](#) on page 313 below.

There’s an alternative to creating several projects for several workspace subtrees: use AccuRev’s include/exclude facility to configure the AccuRev workspace with just the files you want. Then, create an Eclipse project using the entire workspace. See [Working in Include/Exclude Mode](#) on page 67 of the *AccuRev User’s Guide (GUI Edition)*.

## Working in the Navigator View

The Navigator view shows all the files and directories (folders) in your projects. In an AccuRev workspace, objects under version control appear with color arrowhead icons, indicating their AccuRev status.

Icon	Status
yellow, UP-LEFT	overlap
orange, UP	stale
blue, RIGHT	kept
teal green, DOWN	modified
green, UP	backed
gray, RIGHT	member
(none)	external



Often, an element has more than one status flag. In that case, the following precedence order determines which icon appears:

overlap (highest precedence), stale, kept, modified, backed, member (lowest precedence)

## Executing an AccuRev Action

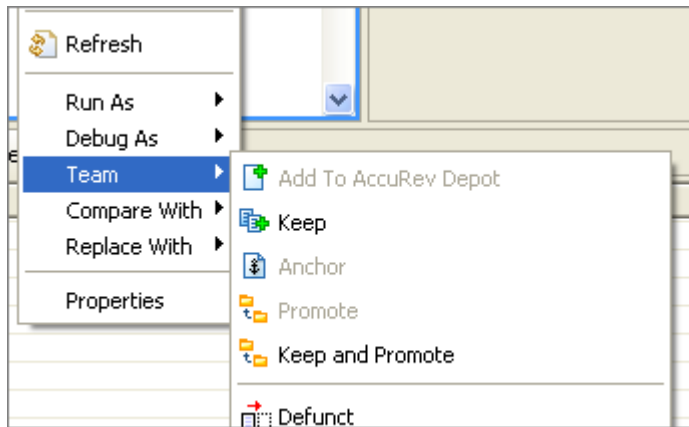
All AccuRev-related “actions” are located on the **Team** submenu of a selection (consisting of files and/or directories) in the Navigator view:

1. Select the files and/or directories to be processed by the action.

Eclipse allows you to make a multiple-object selection containing objects from multiple projects. AccuRev actions will be available only if all the objects are in projects that are “shared” with AccuRev and all the objects belong to the same AccuRev workspace.

2. Right-click the selection to open its context menu.
3. Go to the **Team** submenu, which contains all the available AccuRev actions
4. Select one of the AccuRev actions.





## Output from AccuRev Actions

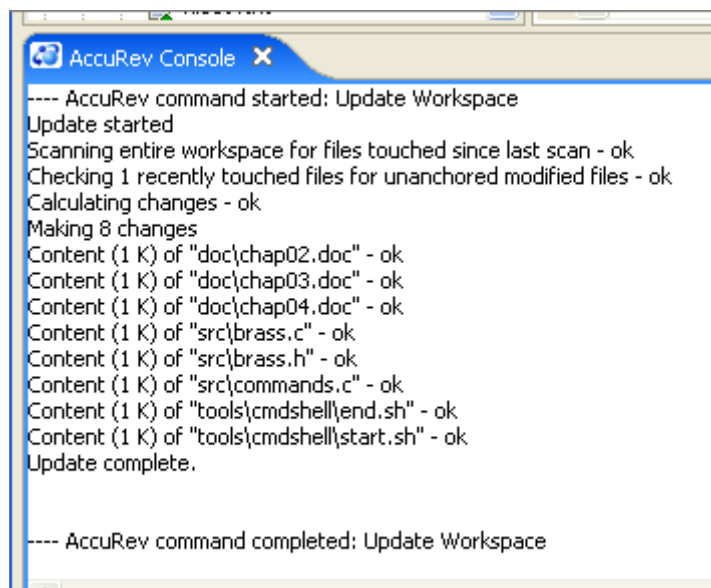
The AccuRev plug-in uses two views to present command output: the AccuRev Console view and the AccuRev Search view.

### AccuRev Console View

Many AccuRev commands generate informational messages, which are displayed in the AccuRev Console view:

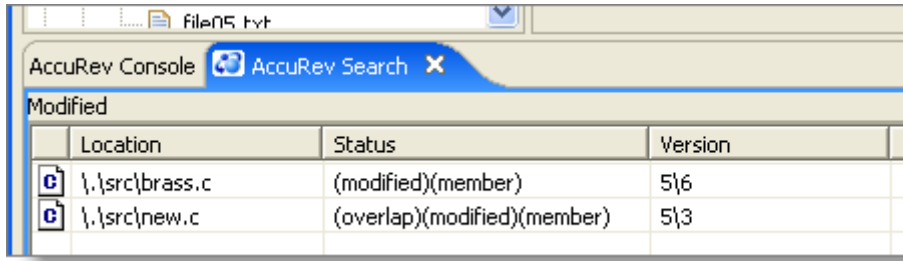
This view automatically appears whenever an AccuRev command produces a user message. You can also make the console appear with the command **Window > Show View > Other**. Select **AccuRev > AccuRev Console** from the window that appears.

Each time you invoke an AccuRev action through the **Team** menu, the contents of the AccuRev Console view are replaced by the command output. In some cases, this view may show the output of multiple AccuRev commands that were generated by a single Team-menu action — for example, **Keep and Promote**.



### AccuRev Search View

When you perform an AccuRev search (for example, **Team > AccuRev Searches > Modified**), the results are displayed as a table in the AccuRev Search view:



The screenshot shows the 'AccuRev Search' console window. It contains a table with the following data:

	Location	Status	Version
	\\.\src\brass.c	(modified)(member)	5\6
	\\.\src\new.c	(overlap)(modified)(member)	5\3

This table works in the usual way:

- Resize a column by dragging its right-hand column separator.
- Resize a column to accommodate the longest value by double-clicking its right-hand column separator.
- (You cannot rearrange the columns.)
- Click on a column header to sort the rows on that column's value. A second consecutive click reverses the order of the rows.

### Searches in Projects that Include Only Part of an AccuRev Workspace

As noted in *Creating an Eclipse Project for Your AccuRev Data* on page 302, you can create an Eclipse project that includes just a *subtree* of an AccuRev workspace. You might create multiple such “subtree” projects that, as a group, contain the entire workspace. On the other hand, you might leave some portion(s) of the AccuRev workspace “uncovered” by the group of Eclipse projects.

When you select one or more files and/or directories and then perform an AccuRev search, the AccuRev Server provides the plug-in with the search results for the entire AccuRev workspace. The AccuRev Console initially filters the search results, displaying only the elements contained in the Eclipse project(s) containing the files and/or directories in your selection. You can use the **Entire AccuRev Workspace** control to remove this filter.

Note: these results can include elements that are not included in any of the Eclipse projects. The Team actions are not available for such elements. This might be a bit confusing!

### Persistence of Data in the AccuRev Views

The data displayed in the AccuRev Console or AccuRev Search view is discarded when you close the view with its “X” control. To preserve the data, minimize the view instead of closing it.

## AccuRev Actions Summary

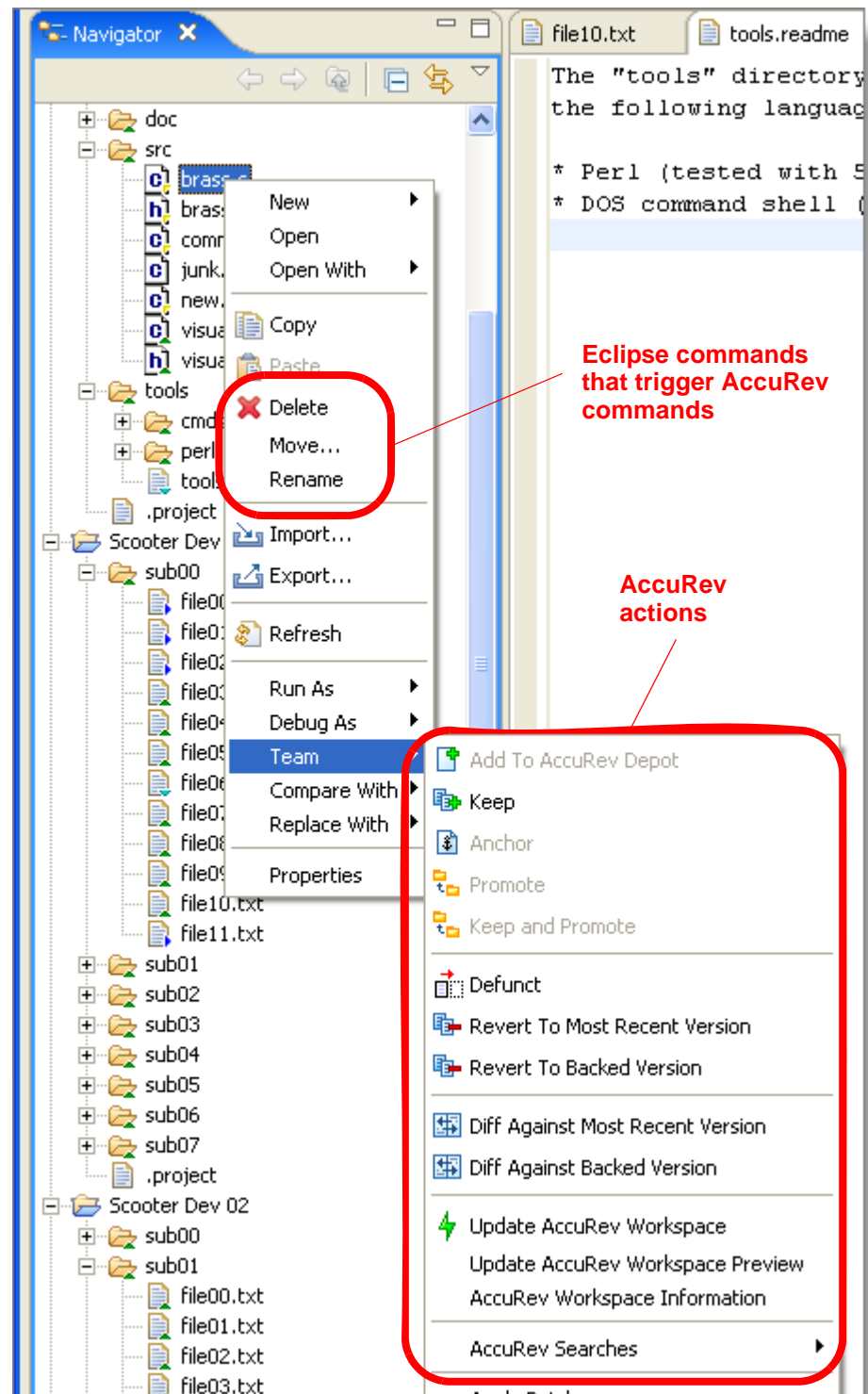
The sections below describe each of the actions available on the Team submenu of a selection in the Navigator view. Note that:

- Not all actions are enabled for a multiple-object selection.
- The results displayed for an AccuRev Search depend on which object(s) are selected when you invoke the search.

In addition to offering these explicit AccuRev actions, the plug-in automatically invokes an AccuRev command when you invoke these Eclipse commands:

- **Rename**
- **Move**

See *Non-Team Commands that Invoke AccuRev Commands* on page 314.



## Add to AccuRev Depot

The **Add to AccuRev Depot** action converts one or more of the files in the development project into AccuRev version-controlled elements. The directory containing the files is also converted to an element, if necessary.

## Keep

The **Keep** action saves the changes you've made to one or more files as “private” versions in the AccuRev repository. These versions are visible only in your workspace — not in the “public” backing stream or in other users' workspaces.

Don't confuse the versions of a file created by **Keep** with the “local history” copies of the file created when you invoke **File > Save** in an Editor pane. Local history copies are maintained by Eclipse itself in the local file system; versions created with **Keep** exist permanently within the AccuRev repository.

## Anchor

Many version control systems have a “check out” command that makes a file writable, so that you can edit it. But AccuRev's default is to have your files *always* be writable. AccuRev *does* have its own similar command, called **Anchor**. This doesn't change the writability of a file, but it does activate the file in your workspace (that is, adds it to the workspace's default group). One effect of this is to ensure that the file won't be overwritten by an **Update AccuRev Workspace** action. Under normal circumstances, you rarely need to invoke the **Anchor** action.

If the AccuRev workspace uses the AccuRev “exclusive file locking” feature, files are initially read-only. Before editing a file, you *must* **Anchor** it, making it writable.

## Promote

The **Promote** action converts one or more “private” versions into “public” versions. That is, it takes versions that you previously created in your workspace with **Keep**, and sends them to the backing stream shared by you and other members of your development team.

Invoking this action can activate either or both of the integrations between AccuRev's configuration management functionality with its issue management (AccuWork) functionality. One of them uses change packages as the point of integration. The other uses a particular issue-record field as the point of integration. Both of them record information about the **Promote** transaction in a user-specified AccuWork issue record. For more information, see [Promote-Based Integrations with Issue Management](#) on page 315.

## Keep and Promote

The **Keep and Promote** action invokes a sequence of AccuRev commands on the selected file(s):

- Performs a **Keep** on all the files in the selection that have changes that have not yet been preserved in the repository. Such files have (**modified**) status.
- Performs a **Promote** on all the files in the selection that now have (**kept**) status.

## Defunct

Deletes a file from your disk, and also marks it as having (**defunct**) status in the AccuRev workspace. You won't see the file in the Navigator view, because the file is no longer on your disk. But you will see the file in certain AccuRev Searches — Defunct, Pending, or Default Group. The file disappears entirely from the AccuRev workspace when you **Promote** the file to the backing stream.

You can also defunct a directory. But before doing so, consult the description of the **defunct** command in the *AccuRev User's Guide (CLI Edition)*.

Note: Eclipse's **Delete** command does not invoke the AccuRev **Defunct** command. It simply removes the file from local disk storage; no change is made to the AccuRev repository. If you invoke this command on an AccuRev-controlled file, its AccuRev status becomes (**missing**).

## Revert to Backed Version

The **Revert to Backed Version** action discards the changes you've made to a file. It restores the version that was in the backing stream at the time of your most recent **Update AccuRev Workspace**. (But if you promoted one or more versions of the element to the backing stream since your most recent **Update**, it restores the most recently promoted version.)

## Revert to Most Recent Version

The **Revert to Most Recent Version** action is similar to **Revert to Backed Version**, but rolls back a file only as far as the private version you recently created with **Keep**. This action is useful when you edit a file, save it with **File > Save**, then decide to throw away the changes.

## Diff Against Backed Version

The **Diff Against Backed Version** action compares your file with the version currently in the workspace's backing stream.

Note: only text files can be compared.

## Diff Against Most Recent Version

The **Diff Against Most Recent Version** action compares your file with the private version you recently created with **Keep**. ("What have I changed since my last **Keep**?")

Note: only text files can be compared.

## Update AccuRev Workspace

The **Update AccuRev Workspace** action copies versions from your workspace's backing stream into your workspace. This has the effect of incorporating other people's changes, which they have promoted to the backing stream, into your workspace.

## Update AccuRev Workspace Preview

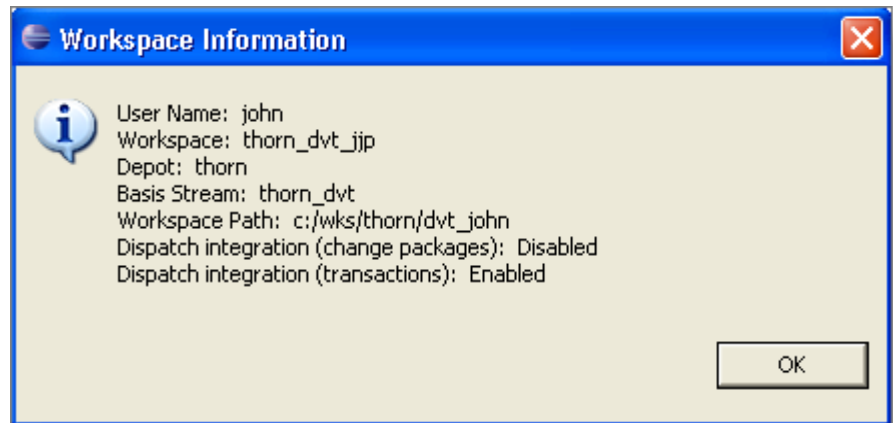
The **Update AccuRev Workspace Preview** action lists in the AccuRev Console that files that *would* be affected by an actual workspace update.

## AccuRev Workspace Information

### The AccuRev Workspace Information

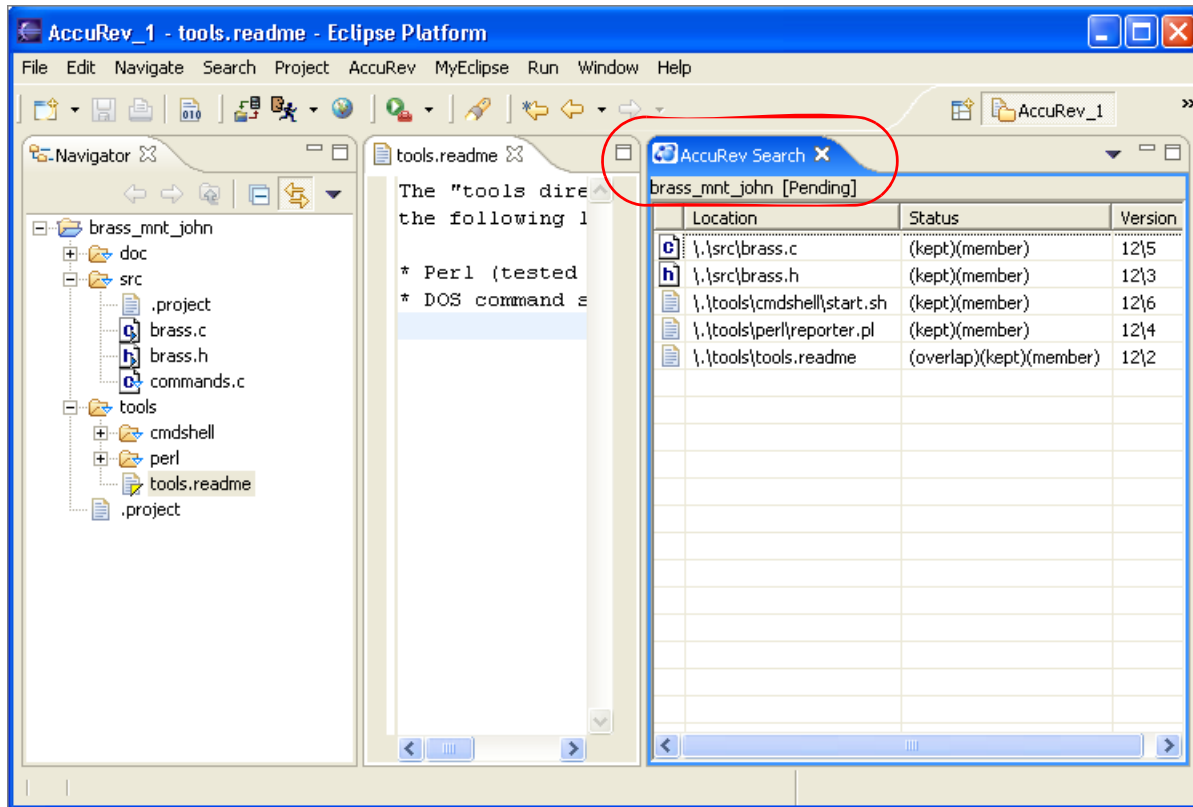
action displays a message box containing information about the current AccuRev context: your user name, your AccuRev workspace name and location, etc. It also indicates whether the integrations between AccuRev's configuration

management and issue management capabilities are enabled. See *Promote-Based Integrations with Issue Management* on page 315.



## AccuRev Searches

The results of an AccuRev search is a table displayed in the AccuRev Search view, showing a selected set of files and/or directories. The previous contents of the AccuRev Search view, if any, are discarded.

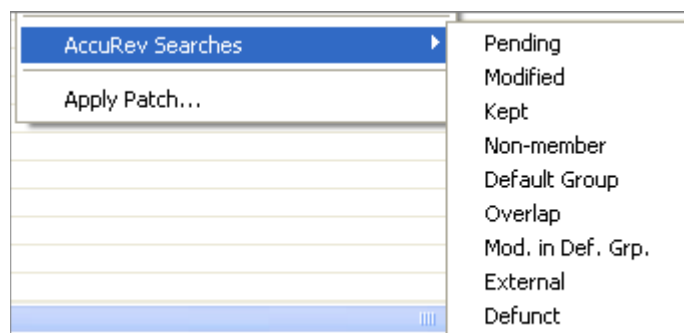


A search operates on the entire AccuRev workspace, so each object located by a search is listed by its pathname relative to the workspace’s top-level directory. (This is called a “depot-relative” pathname in AccuRev.)

Note: the AccuRev Search view sometimes initially displays a subset of the objects located by a search. See *Searches on Parts of a Workspace* on page 313.

The AccuRev plug-in implements most of the searches available in the AccuRev GUI.

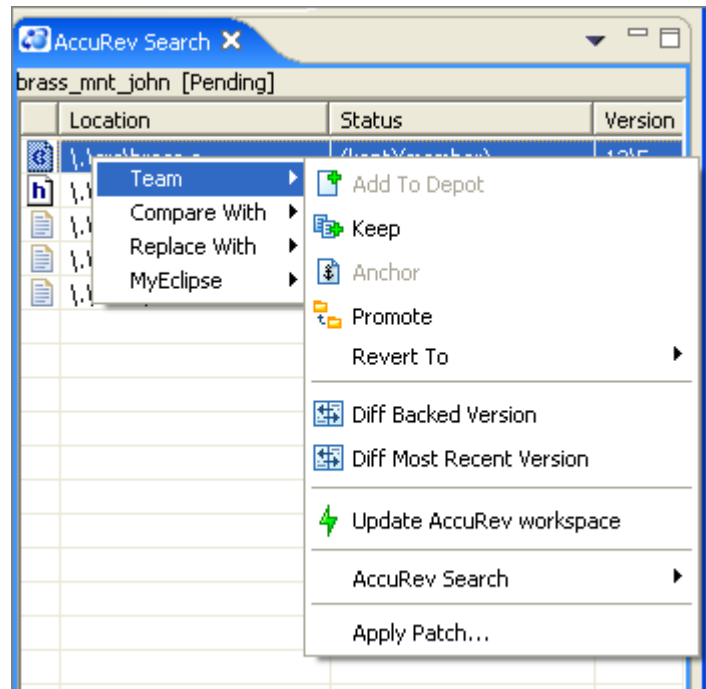
- Pending
- Modified
- Kept
- Non-member
- Default Group
- Overlap
- Modified in Default Group
- External
- Defunct



(The searches not implemented are Deep Overlap, Missing, and Stranded.) For a full discussion of searches see *AccuRev File Statuses* on page 48 of the *AccuRev User’s Guide (GUI Edition)*.

You can invoke AccuRev actions on the items displayed in the AccuRev Search view, using their context menu (**Team** submenu). This makes it easy to perform such operations as:

- ...promoting some or all of the elements that are pending promotion
- ... placing under version control some or all of the workspace's external files



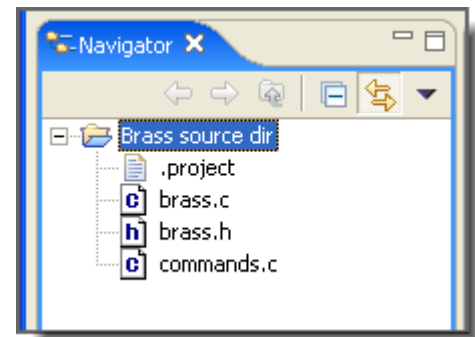
### Searches on Parts of a Workspace

As described in *Creating an Eclipse Project for Your AccuRev Data* on page 302, you can create an Eclipse project that contains a particular subtree of an existing AccuRev workspace. For such a project, the AccuRev Search view can display either full or partial results of a workspace search:

- By default, the AccuRev Search view filters the search results, displaying only elements that are in the current Eclipse project. (... or projects — see the note below)
- You can use the drop-down menu near the minimize and maximize controls to check the **Entire AccuRev Workspace** option. This removes the filtering of search output — elements that meet the search criterion throughout the workspace are displayed, even if they are not in the current Eclipse project.

Use the drop-down menu to toggle the **Entire AccuRev Workspace** setting. This setting persists as long the AccuRev Searches view remains open.

For example, suppose you've loaded just a workspace's **src** directory — not the **doc** or **tools** directory — into an Eclipse project. Invoking the action **Team > AccuRev Searches > Pending** displays the elements in the Eclipse project (that is, in the **src** directory) that are pending promotion. Then, turning on the **Entire AccuRev Workspace** setting expands the display to include pending-promotion elements in the AccuRev workspace that are not loaded into the Eclipse project.



Note: in the description above, the “current Eclipse project” is the one whose objects were selected when you opened the **Team** context menu. You can select elements from multiple Eclipse projects and invoke an AccuRev search, as



long as all the elements belong to the same AccuRev workspace. In this case, the AccuRev Search view initially filters the results to include only the elements from the set of projects whose elements were selected.

Brass source dir [Pending]

	Location	Status	Version
	\\.\src\brass.c	(kept)(member)	12\5
	\\.\src\brass.h	(kept)(member)	12\3

elements in Eclipse project (SRC directory) that satisfy search criterion

Brass source dir [Pending]

	Location	Status	Version
	\\.\src\brass.c	(kept)(member)	12\5
	\\.\src\brass.h	(kept)(member)	12\3

Show all elements

Brass source dir [Pending]

	Location	Status	Version
	\\.\src\brass.c	(kept)(member)	12\5
	\\.\src\brass.h	(kept)(member)	12\3
	\\.\tools\cmdshell\start.sh	(kept)(member)	12\6
	\\.\tools\perl\reporter.pl	(kept)(member)	12\4
	\\.\tools\tools.readme	(overlap)(kept)(member)	12\2

elements in entire AccuRev workspace that satisfy search criterion

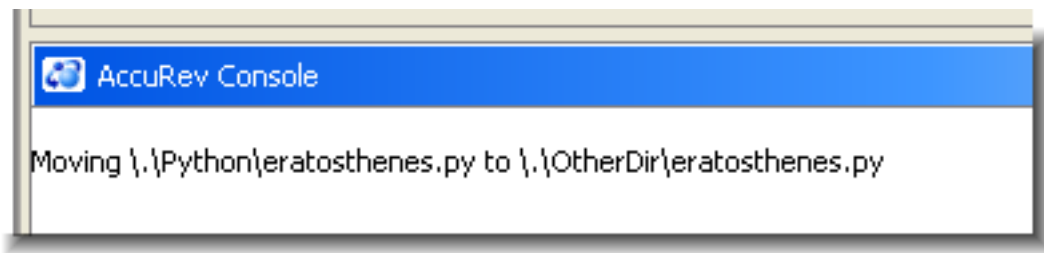
## Non-Team Commands that Invoke AccuRev Commands

Eclipse provides a number of methods for changing the pathname of a file or directory:

- Renaming an object:
  - Right-click the object, and select **Rename** from the context menu.
  - Select the object, and press function key **F2**.
- Moving an object to a different directory:
  - Right-click the object, and select **Move** from the context menu.
  - Drag-and-drop the object from one directory to another.

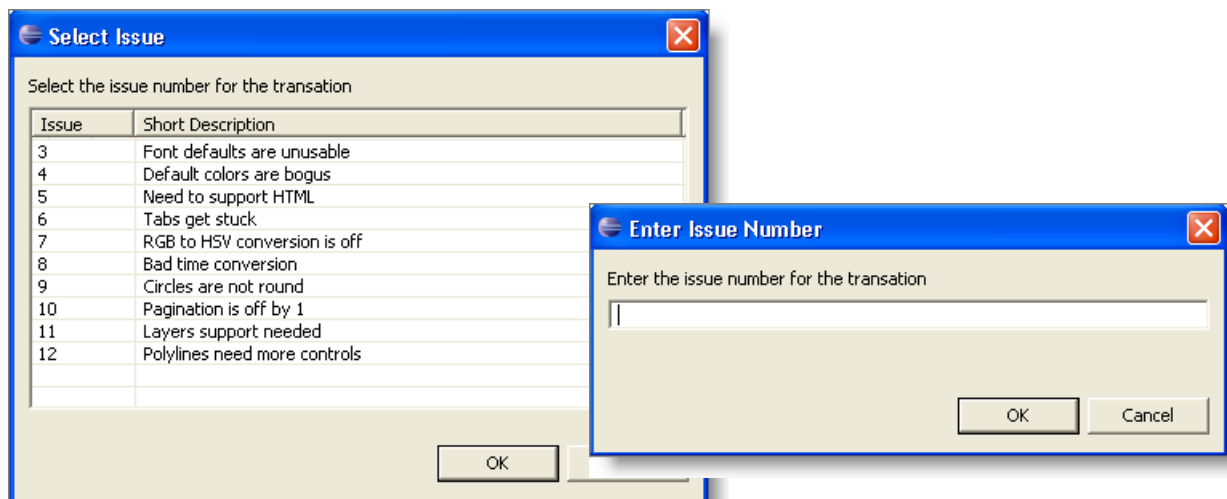
Eclipse enforces the restriction that you cannot move an object to a different Eclipse project (AccuRev workspace). This corresponds to AccuRev's own restriction in this area.

When you complete any of the above actions, the plug-in automatically invokes the AccuRev **Rename** command, to record the change of pathname in the repository. As usual, the results are displayed in the AccuRev Console view:



## Promote-Based Integrations with Issue Management

AccuRev defines two integrations — change-package-based and transaction-based — between its configuration management functionality and its issue management (AccuWork) functionality. The integrations are enabled separately, through AccuRev commands that are not available through Eclipse. One or both of the integrations is triggered when you invoke a **Promote** action on a set of elements. You are prompted to specify a AccuWork issue record in a pop-up window.



If only the transaction-based integration is enabled *and* you do not have a default query defined for the issues database, the prompt window contains a text field into which you type an issue number. Otherwise, the prompt window offers a set of existing issue records, from which you select one. Then:

- The change-package-based integration (if enabled) records the promoted versions on the Changes tab of the specified issue record.
- The transaction-based integration (if enabled) records the **Promote** transaction number in the **affectedFiles** field of the specified issue record.

For more information, see *Integrations Between Configuration Management and Issue Management* on page 220 of the *AccuWork Issue Management Manual*.



# Using AccuRev with SCC-Compliant Applications

This chapter describes the integration of AccuRev with Windows applications that support the Microsoft SCC (Source Code Control) programming interface. The SCC interface enables users of the application to access AccuRev version-control facilities using its own menu structure.

As of AccuRev Version 4.0, this integration has been designed for, and validated with, the following:

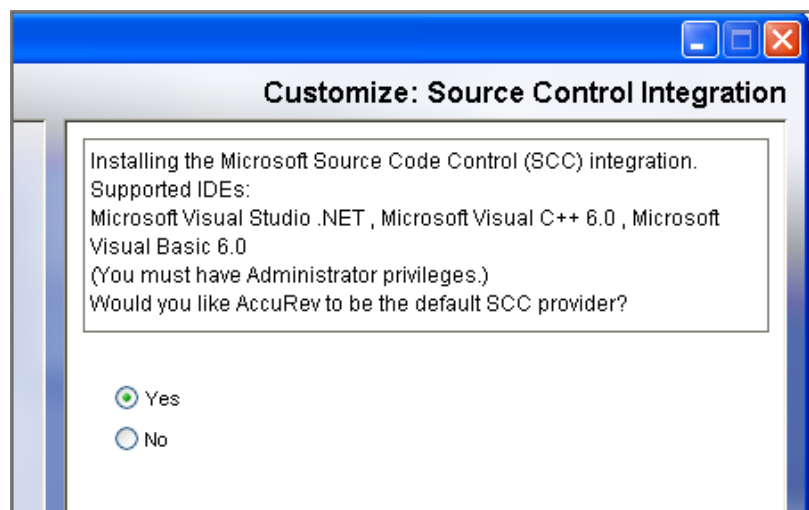
- Microsoft Visual Basic 6.0
- Microsoft Visual C++ 6.0
- Microsoft Visual Studio .NET

Check file **SCCAcc-ReadMe.txt** (located in the AccuRev **bin** directory) for updated information on which applications the integration supports.

## Installing the Integration

Installation of AccuRev on a Windows machine automatically includes the AccuRev-SCC integration. The installation wizard has you choose whether AccuRev should be the default provider of SCC services.

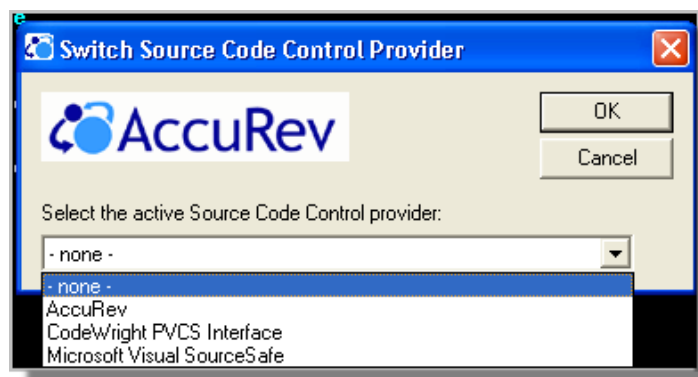
Before running the AccuRev installation program, make sure to exit all the tool(s) on your machine that use the SCC interface.



## Turning Off the Integration

There is no way to uninstall the AccuRev-SCC integration, except to uninstall AccuRev itself. But you can turn off the integration using the utility program **SwitchSCC.exe**, which is located in the AccuRev **bin** directory.

This program displays a simple dialog box. You can disable the SCC programming interface completely (select **-none-**), or you can enable



another SCC provider. Only one SCC provider can be active at a time, so selecting another provider also turns off the AccuRev-SCC integration.

## Using a Workspace

The various SCC commands move data between the central AccuRev repository and your personal work area. Your work area must be located in an existing AccuRev workspace. You must create this workspace with the AccuRev GUI or CLI; there is no SCC command to create a workspace.

Note: if the workspace containing a project becomes inaccessible (for example, if it was deactivated with the AccuRev **remove** command), SCC commands will not work for that project. You might need to delete and recreate the project file. And you might need to restart the application before using SCC commands with projects located in other workspaces.

## Establishing Your Identity

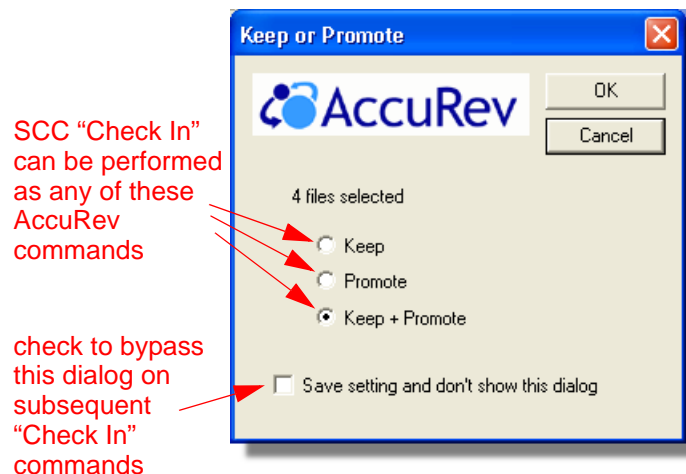
Most AccuRev commands, invoked through the AccuRev-SCC integration, must be performed by an official user. That is, you must have an AccuRev username (or “principal-name”). The integration uses the identity most recently established by the AccuRev GUI. This setting is stored in your AccuRev preferences file, **preferences.xml**, located in the **.accurev** subdirectory of your home directory.

## SCC Commands and AccuRev Commands

The SCC programming interface enables a predetermined set of source-code-control commands: **Add to Source Control**, **Check Out**, **Check In**, etc. When you invoke an SCC command (say, **Check In**), control is handed off to the corresponding AccuRev command.

The SCC model of source-code-control functionality does not match the AccuRev model exactly. For example, SCC’s **Check In** command does not correspond exactly to a single AccuRev command. Accordingly, when you invoke **Check In**, AccuRev must determine exactly how to perform this operation. You can make a selection each time from a dialog box displayed by the integration.

But for convenience, you’ll probably want to configure the integration to remember how the SCC command is to be executed by AccuRev. The AccuRev-specific dialog box won’t appear on subsequent invocations of the SCC command. You can configure the integration either by selecting a checkbox in the dialog box for a specific command (shown above), or by using the overall AccuRev Plug-In Options dialog (see *Plug-In Options* on page 327).



## AccuRev Command Output

AccuRev commands generate confirmation messages, which are displayed within the application:

- Visual Basic: the messages are displayed in a separate AccuRev Messages window. (Bugs in Visual Basic's implementation of the Source Control Results Window prevent the use of that window.)
- Visual C++: the messages are displayed in the Source Control tab of the Output window.
- Visual Studio .NET: the integration displays messages in the Source Control tab of the Output window, and AccuRev displays its own messages in a separate AccuRev Messages window.

## SCC Command Summary

The sections below describe each of the SCC commands, differences (if any) between the SCC and AccuRev models, and the AccuRev-specific dialog boxes that map the SCC commands to AccuRev commands.

Note: the **Open From Source Control** and **Add Project From Source Control** SCC commands, displayed on the **File > Source Control** menu in Visual Studio .NET, are not supported.

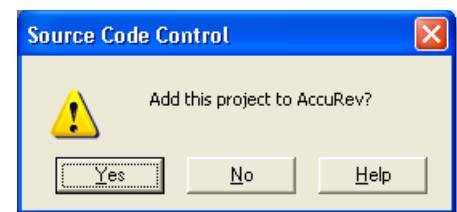
### Add to Source Control

The SCC **Add to Source Control** command converts one or more of the files in the development project into AccuRev version-controlled elements. (The directory containing the files is also converted to an element.) The files to be converted must be located in a directory within an existing AccuRev workspace.

Some IDEs can be configured to invoke this command automatically for a new project, or for a new file added to an existing project. Check the application's Source Code Control options for the available configuration options.

You can use an explicit command to place an entire project or individual file, under source control:

- Visual Basic: **Tools > AccuRev > Add Project to AccuRev**
- Visual C++: **Project > Source Control > Add to Source Control**
- Visual Studio .NET: **File > Source Control > Add Solution to Source Control** (or **All Selected Projects to Source Control**)



The integration implements this command in either of these ways:

- Perform an **add** command, creating an initial version of each file in your workspace stream.
- Perform an **add** command, as above, and also perform a **promote** command to propagate the initial version of the file(s) to the backing stream.

Note: when Visual Basic adds a new project to source control, it handles the project file (**.vbproj**) separately from the other files. Accordingly, if the AccuRev feature described in *Promote-Based Integrations with Issue Management* on page 330 is enabled, you'll be prompted twice to specify an issue number. Be sure to enter the same number both times!

## Check Out

The SCC **Check Out** command is intended to make a file writable, so that you can edit it. By default, though, files in an AccuRev workspace are *always* writable, so you don't need to perform **Check Out** commands when working in the application. (For a discussion of the non-default case, see *Working in an Exclusive File Locking Workspace* below.)

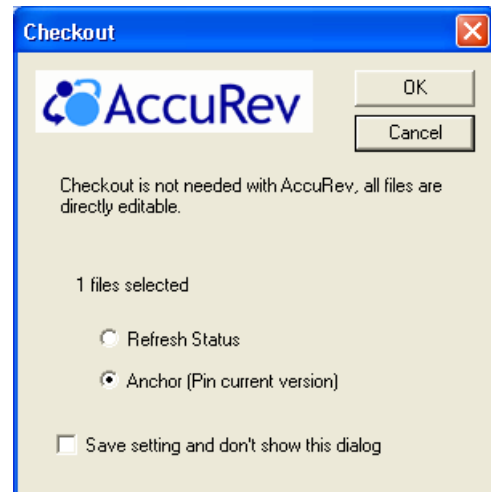
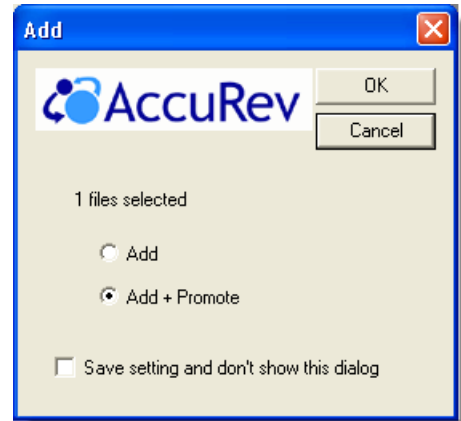
AccuRev does have its own “check out” command — called **co** (or equivalently, **anchor**). In a standard workspace, this doesn't change the writability of a file, but it does activate the file in your workspace (that is, adds it to the workspace's default group). One effect of this is to ensure that the file won't be overwritten by a **Get Latest Version** command (when it performs an AccuRev **update** command). Under normal circumstances, you rarely need to invoke the **co** command.

Accordingly, when the integration performs a **Check Out**, there are two choices:

- Have the application refresh the icon annotation indicating the file's status. No change takes place at the AccuRev level.
- Perform an AccuRev **co (anchor)** command, activating the file in your workspace.

Note: you can also refresh the icon annotation with a command:

- Visual Basic: **Tools > AccuRev > Refresh File Status**
- Visual C++: **Project > Source Control > Refresh Status**
- Visual Studio .NET: **File > Source Control > Refresh Status**



These commands examine all source files, and turn on the “checked out” icon annotation for each “active” file. This includes files whose contents you have modified, files on which you’ve run the AccuRev **Keep** command (see *Check In* below), and any other files that are members of the AccuRev workspace’s default group — for example, renamed files.

## Working in an Exclusive File Locking Workspace

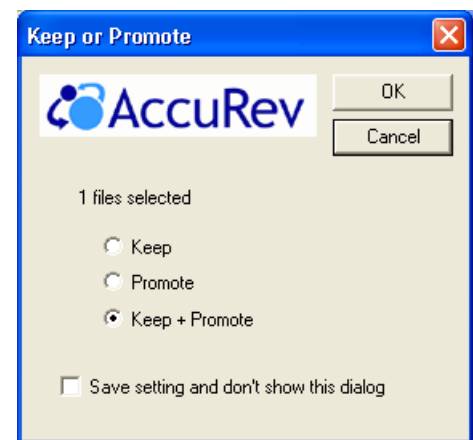
An AccuRev workspace can be configured to use the exclusive file locking feature: version-controlled files are read-only in such a workspace, until you use the **Check Out** command to make them writable.

## Check In

The SCC **Check In** command saves the changes you’ve made to one or more files in the source-code repository. AccuRev’s “two level check-in” model means you have the choice of keeping the changes private (in your workspace) or making them public (in the backing stream):

- Perform a **keep** command, creating a new version of each file in your workspace stream.
- Perform a **promote** command, propagating versions previously created with **keep** to the backing stream.
- Perform both **keep** and **promote** commands at the same time.

Note: the application removes the “checked out” icon from a file only when you promote it to the backing stream. If you execute the **keep**-only variant of the **Check In** command, the application still considers the file to be checked out.



## Undo Check Out

The SCC **Undo Checkout** command is intended to discard the changes you’ve made to a file after your most recent **Check Out**.



The integration implements this command in either of these ways:

- Discard all the changes you've made since the last time you performed an AccuRev **promote** of the file (with **Check In**). Note that this might discard several intermediate **keep** versions that you created with **Check In**.
- Discard just the changes you've made since the last time you performed an AccuRev **keep** of the file (with **Check In**).



## Get Latest Version

The SCC **Get Latest Version** command is intended to copy the most recent version of a file from the central AccuRev repository to your workspace. AccuRev's **Update** command does not operate on individual files, though — it updates the entire contents of your workspace. That might be more than you bargained for!

Accordingly, the **Get Latest Version** command offers a choice:

- Run AccuRev's **Populate** command only. This downloads from the repository the version you previously brought into your workspace with an **Update**. (But if you had subsequently preserved one or more versions with the **Keep** command — see [Check In](#) above — **Populate** brings in the latest such version instead.)
- First run AccuRev's **Update** command, then run **Populate**. This downloads versions that were created by other users since your previous **Update**.



With either choice, the **Populate selected files only** checkbox restricts the operation of **Populate** to the currently selected files. This is useful when you need to restore one or more files that were inadvertently deleted. With this checkbox cleared, **Populate** makes sure that all version-controlled files in the workspace are at the correct version level — as of the previous update (**Populate** option), or as of the update that just completed (**Update and Populate** option).

Note: **Populate** is careful — it never overwrites a file that you have saved with **Keep** but not yet **Promote**'d. And it won't overwrite a file that you have edited, but not yet saved with **Keep**.

The **More Options** button doesn't update any files at all — it brings up a dialog that provides access to a variety of AccuRev commands. See [Additional AccuRev-Related Commands](#) on page 325.

When you open an existing source-controlled project in some applications, it offers to run the **Get Latest Version** command. Keep in mind that the resulting AccuRev (**Populate** and) **Update** commands will affect the entire AccuRev workspace, not just the files in that particular project.

## Show Differences / Compare Versions

The SCC **Show Differences** or **Compare Versions** command compares your file with an older version in the repository:

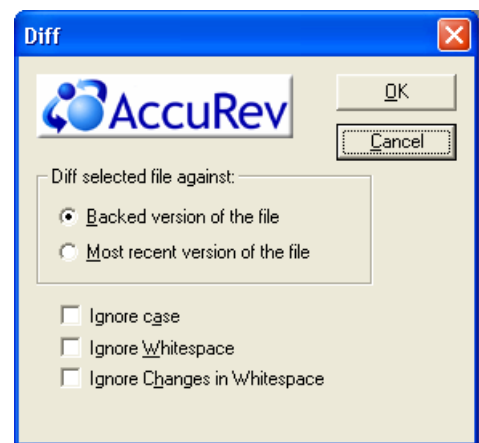
- Visual Basic: **Tools > AccuRev > Show Differences**
- Visual C++: **Project > Source Control > Show Differences**
- Visual Studio .NET: **File > Source Control > Compare Versions** or file's context menu

Note that only text files can be compared with this command.

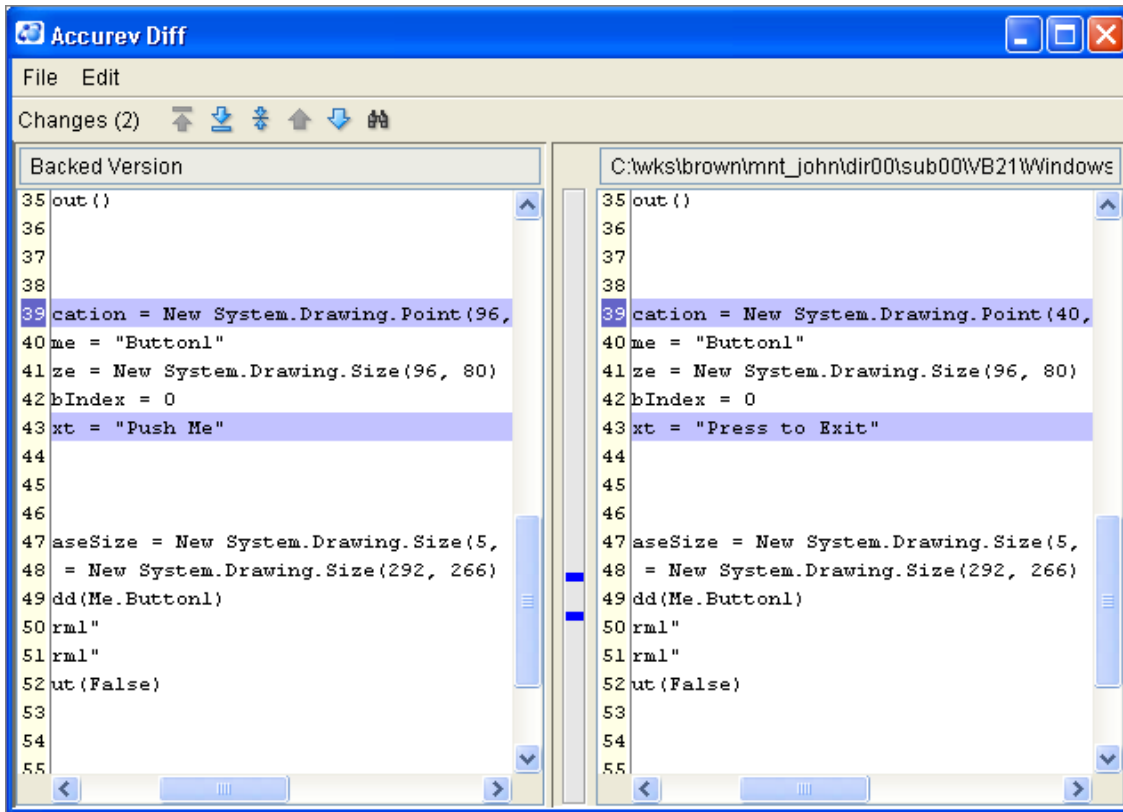
The integration implements this command in either of these ways:

- Compare with the “public” version in the backing stream.
- Compare with the most recent “private” version you created in your workspace.

Either way, you have several options regarding the way case and whitespace are handled.



The differences between the two versions are displayed in a separate window, with the AccuRev Diff tool:



## Show History

The SCC **Show History** command displays the set of transactions involving the selected file. This command is located on the main menu (not on a file's context menu):

- Visual Basic: **Tools > AccuRev > Show History**. The history listing is sent to the AccuRev Messages window.
- Visual C++: **Project > Source Control > Show History**. The history listing is sent to the Source Control tab of the Output pane.
- Visual Studio .NET: **File > Source Control > History**. The history listing is sent to the AccuRev Messages window.

## Save As

The application commands **Save As** and **Save Project As** create a new file. In Visual Basic, these commands trigger an SCC name-change operation, changing the name of the file in the repository. For example, if you select **Proj21.frm** and Save As to **widget\_project.frm**, AccuRev uses the **move** command to rename file element **Proj21.frm** to **widget\_project.frm**. The file **Proj21.frm** remains in your workspace, as a file not under source control (an “external” file, in AccuRev terms).

The integration implements this in either of these ways:

- Perform an AccuRev **move** command,
- Perform an AccuRev **move** command, followed by a **promote** of the element.

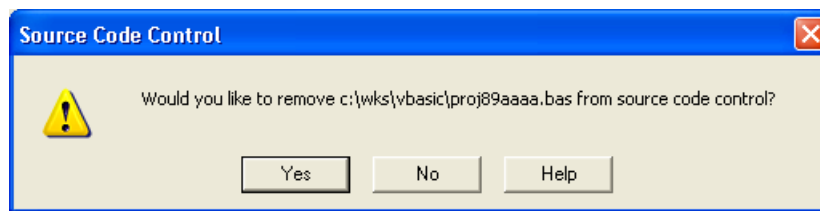
Visual Studio .NET has a **Rename** command (for files, projects, and solutions), but this command is not accompanied by an AccuRev **move** command. To avoid confusion, avoid using the Visual Studio .NET **Rename** command.

VC++ does not have a **Save As** or **Rename** command.



## Remove

The Visual Basic command **Remove <filename>** removes a file from a project, but leaves the file on your disk. Visual Basic asks whether you also want to remove the file from the source-control system:

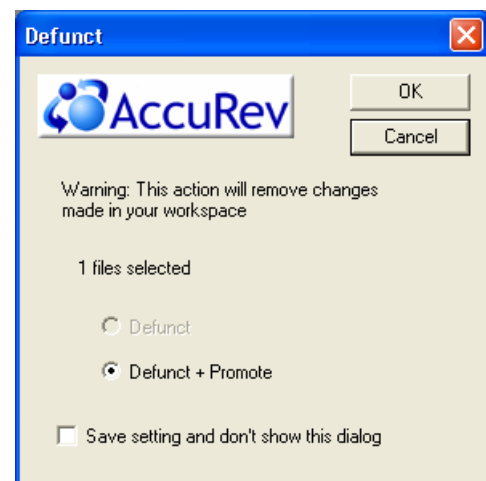


If you select **Yes**, the integration implements this in either of these ways:

- Perform an AccuRev **defunct** command (currently disabled).
- Perform an AccuRev **defunct** command, followed by a **promote** of the element.

Note: the **defunct** command removes the file from your disk, but the integration immediately restores it, to conform with the **Remove** specification.

In Visual C++, the command **Project > Source Control > Remove from Source Control** brings up the same AccuRev dialog.



Visual Studio .NET has a **Remove** command for projects and a **Delete** command for files. But these commands are not accompanied by AccuRev **defunct** commands. The data structure(s) disappear from the application, but remain under AccuRev source control.

## Additional AccuRev-Related Commands

The AccuRev-SCC integration provides additional commands through this dialog box:



To access this dialog box ...

- Visual Basic: **Tools > Accurev.**
- Visual C++: **Project > Source Control.**
- Visual Studio .NET: **File > Source Control > AccuRev** (or **Get Latest Version > More Options** on a file's context menu)

The commands are described in the following sections.

## Run AccuRev

Starts the AccuRev GUI, enabling you to perform operations that are not supported by the AccuRev-SCC integration.

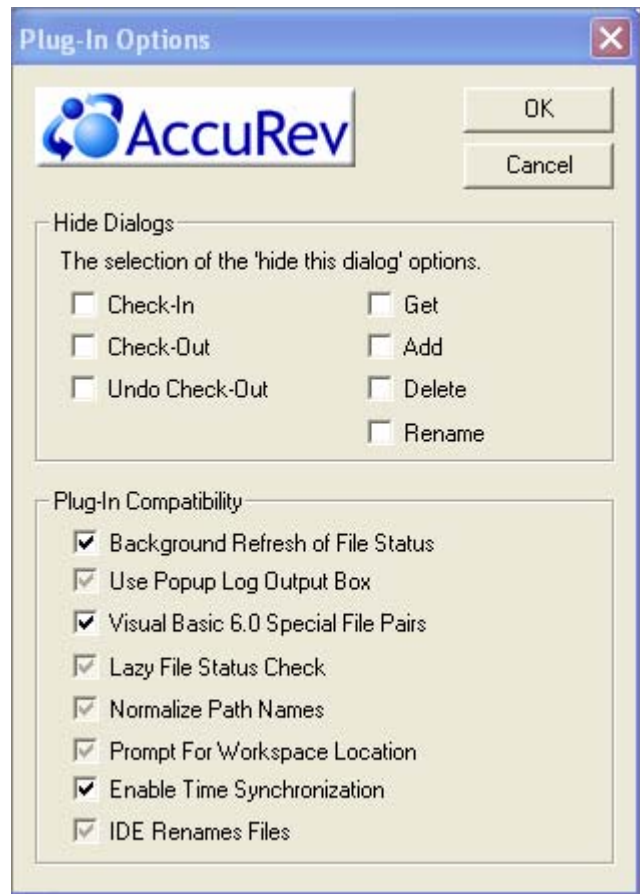
## Plug-In Options

Displays a dialog with which you can control, on a command-by-command basis, the level of interactivity of AccuRev commands invoked through the SCC interface.

Use the various checkboxes in the **Hide Dialogs** section to suppress the AccuRev dialogs invoked by particular SCC commands; the most recent selection you made in the dialog will be used automatically.

The checkboxes in the **Plug-In Compatibility** section control various aspects of AccuRev-SCC integration functionality.

Note: these are tri-state checkboxes: the feature is either enabled (box is checked), disabled (box is cleared), or defaulted (box has gray check). “Defaulted” means to use the application’s default for the feature. (A particular application might not support the feature at all.)



### Background Refresh of File Status

Controls periodic, automatic execution of the Refresh Status operation (see above).

### Use Popup Log Output Box

Controls use of an AccuRev-specific log window (“AccuRev Messages”), instead of the application’s own log window or tab. A change in this setting takes effect the next time you start the application.

### Visual Basic 6.0 Special File Pairs

Controls a facility in which certain files hidden by Visual Basic’s Project Explorer window are automatically processed at the same time as their visible “partner” files. For example, when you **Check In** the visible file **widgets.frm**, the hidden file **widgets.frx** will be checked in, too. This facility is initially enabled. The file pairs handled by this facility are:

Visible File Suffix	Hidden File Suffix
.frm	.frx
.dob	.dox
.ctl	.ctx

Visible File Suffix	Hidden File Suffix
.pag	.pgx

### Lazy File Status Check

Schedule automatic execution of the Refresh Status operation to occur less frequently (in batches, rather than for individual files). This improves performance, but increases the possibility that a file's status can be displayed incorrectly for a short time.

### Normalize Path Names

(enable only if the depot is case-sensitive) Causes the integration to determine the proper case of each filename passed to it by the application.

### Prompt for Workspace Location

Enables a dialog in which you are prompted for the disk location of your workspace (in cases where the application doesn't supply this information).

### Enable Time Synchronization

Enables/disables the **Synchronize Time** button.

### IDE Renames Files

Tells the integration that the application itself can rename files at the operating-system level (checked), or tells the integration to do the renaming itself (not checked). This enables the AccuRev **move** command to proceed correctly, keeping a file's operating-system name in sync with its name in the AccuRev repository.

## Synchronize Time

Runs the AccuRev **sync** command, to synchronize the system clock on your machine with the clock on the AccuRev Server machine.

## Update

Performs an AccuRev **Update** command on the workspace.

## Update Preview

Displays the elements that would be changed by a workspace **Update**.

## Workspace Information

Displays information about your processing environment (AccuRev **info** command).

## Populate Missing Files

(see Note 1 below) Restores files that have been deleted or mistakenly changed, by copying them from the AccuRev repository. For each element, the version copied is the one that was in the backing stream at the time of your most recent **Update**. (But if you subsequently created one or more versions with **Keep**, it restores the most recent such version.)

## Merge Overlapping Files

(see Note 1 below) Use AccuRev's graphical Merge tool to combine your changes with changes that a colleague has already **Promote**'d to the backing stream.

## Keep Modified Files

(see Note 1 below) Perform a **Keep** command on files with changes that you have not saved in the AccuRev repository.

## Promote Kept Files

(see Note 1 below) Perform a **Promote** command on files that you have saved with **Keep**, but have not yet promoted to the backing stream.

## Searches / Show

(see Note 1 below) Performs a search through the entire workspace, locating elements with the specified AccuRev status.

## AccuRev Properties

(see Note 2 below) Sends the selected file's AccuRev status (**stat** command) to the messages/output window.

## Refresh Status

(see Note 2 below) If you have modified the selected file without entering a **Check Out** command, redisplay the file with a "checked out" icon annotation. This operation is also available through the dialog box displayed by the **Check Out** command.

Note 1: This command is affected by the **Use Selected Files Only** checkbox:

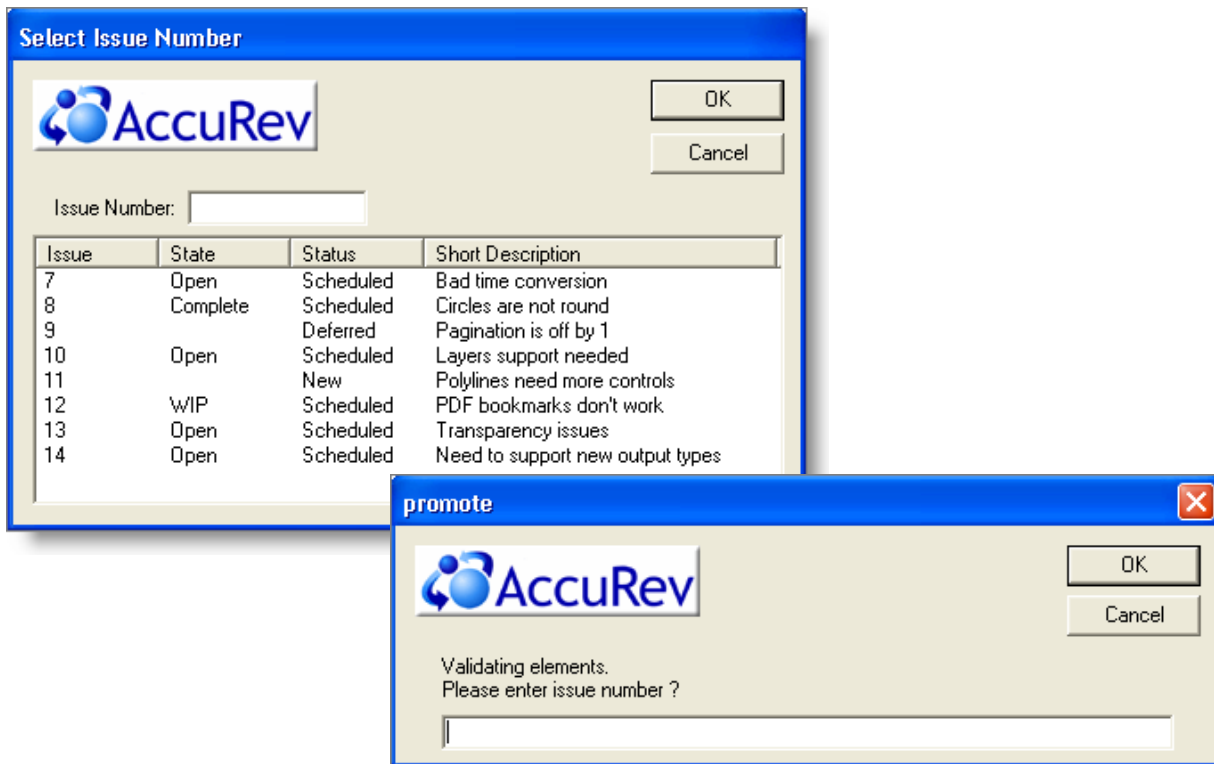
- If the box is checked, the command affects, or reports on, just the currently selected files.
- If the box is not checked, the command affects, or reports on, files throughout the workspace with the specified status.

Note 2: this command is located on the Visual Basic **Tools > AccuRev** submenu.



## Promote-Based Integrations with Issue Management

AccuRev defines two integrations — change-package-based and transaction-based — between its configuration management functionality and its issue management (AccuWork) functionality. The integrations are enabled separately, through AccuRev commands that are not available through the SCC integration. One or both of the integrations is triggered when you invoke a **Promote** action on a set of elements. You are prompted to specify a AccuWork issue record in a pop-up window.



If only the transaction-based integration is enabled *and* you do not have a default query defined for the issues database, the prompt window contains a text field into which you type an issue number. Otherwise, the prompt window offers a set of existing issue records, from which you select one. Then:

- The change-package-based integration (if enabled) records the promoted versions on the Changes tab of the specified issue record.
- The transaction-based integration (if enabled) records the **Promote** transaction number in the **affectedFiles** field of the specified issue record.

For more information, see *Integrations Between Configuration Management and Issue Management* on page 220 of the *AccuWork Issue Management Manual*.

## Notes on Application Usage

The following sections present some notes and suggestions regarding use of the various IDEs and their integrations with AccuRev.

## File Status Icons

The supported IDEs indicate the source-control status of files with special icons:

	Visual Basic 6.0	Visual C++ 6.0	Visual Studio .NET
file not under source control	normal icon	white file icon	auxiliary red-checkmark icon to left of normal icon (pending checkin)
file under source control and checked-in	auxiliary file icon to left of normal icon	gray file icon	auxiliary blue-lock icon to left of normal icon
file under source control and read-only	auxiliary blue-lock icon to left of normal icon		
file under source control and checked-out	auxiliary red-checkmark icon to left of normal icon	red checkmark to left of gray icon	auxiliary red-checkmark icon to left of normal icon

## Files that Should Not Be Placed under Source Control

The IDEs create and manage various auxiliary files, which should not normally be placed under source control:

- Visual Basic 6.0: files with this suffix: **.dca**
- Visual C++ 6.0: files with these suffixes: **.ncb, .clw, .opt**
- Visual Studio .NET: files with this suffix: **.svo**



# Using AccuRev with Sun Java Studio

This white paper describes the integration of AccuRev with the Java-language IDEs NetBeans and Sun Java Studio. (Sun Java Studio, based on NetBeans, was formerly named “Sun ONE Studio” and “Forte for Java”.) The integration enables users of the IDE to access AccuRev version-control facilities using the IDE’s own “Versioning” menu.

As of AccuRev Version 4.0, this integration has been validated with:

- NetBeans 3.5
- Sun Java Studio Standard, Version 5

Check the AccuRev Release Notes for updated information on which IDEs the integration supports.

## Installing the JAR File

The AccuRev Integration is implemented as a single JAR (Java archive) file, named **ac\_sunjava.jar**. The AccuRev installer places this file in the AccuRev **bin** directory.

## Enabling the Integration

Use this procedure to enable the AccuRev Integration in the Java IDE:

1. Start the Java IDE.
2. Select **Tools > Options** from the IDE menu.
3. In the Options window, navigate to **Options > IDE Configuration > System > Modules**.
4. Right-click on **Modules**, and select **Add > Module**.
5. In the file-selection dialog, navigate to the AccuRev **bin** folder, and select **ac\_sunjava.jar**.

## Disabling the Integration

If you wish to disable use of the AccuRev Integration in the Java IDE:

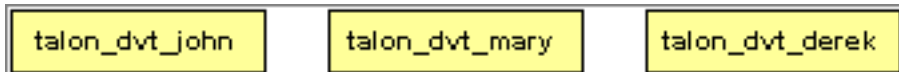
1. Select **Tools > Options** from the IDE menu.
2. In the Options window, navigate to **Options > IDE Configuration > System > Modules > Version Control > AccuRev**.
3. In the right side of the Options window, find the **Enabled** property, whose current value is **True**.
4. Click the **True** value, and use the list-box functionality to change the value to **False**.

## The AccuRev Usage Model

AccuRev's flexibility makes it easy to use for a variety of development scenarios. But like every software system, AccuRev has usage models that were foremost in the minds of its architects. This section describes the most common usage model.

AccuRev is a software configuration management (SCM) system, designed for use by a team of people (users) who are developing a set of files. This set of files might contain source code in any programming language, images, technical and marketing documents, audio/video tracks, etc. The files — and the directories in which the files reside — are said to be “version-controlled” or “under source control”.

For maximum productivity, the team's users must be able to work independently of each other — sometimes for just a few hours or days, other times for many weeks. Accordingly, each user has his own private copy of all the version-controlled files. The private copies are stored on the user's own machine (or perhaps in the user's private area on a public machine), in a directory tree called a workspace. We can picture the independent workspaces for a three-user team as follows:



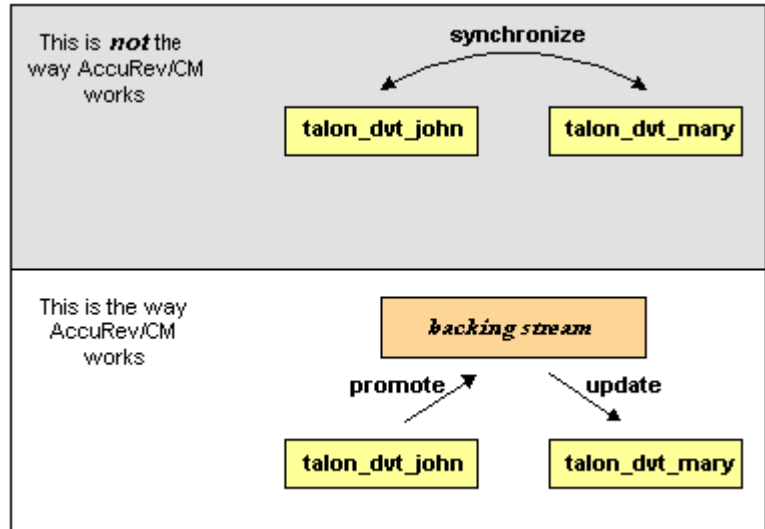
This set of users' workspaces uses the convention of having like names, suffixed with the individual usernames. AccuRev enforces this username-suffix convention. **talon\_dvt** might mean “development work on the Talon product”; **john**, **mary**, and **derek** would be the users' operating system login names.

From AccuRev's perspective, development work in this set of workspaces is a continual back-and-forth between “getting in sync” and “getting out of sync”:

- Initially, the workspaces are completely synchronized: they all have copies of the same set of version-controlled files.
- The workspaces lose synchronization as each user makes changes to some of the files.
- Periodically, users share their changes with each other. When **john** incorporates some or all of **mary**'s changes into his workspace, their two workspaces become more closely (perhaps completely) synchronized.

You might assume that the workspace synchronization process involves the direct transfer of data from one workspace to another. But this is not the way AccuRev organizes the work environment. Instead of transferring data directly between private areas (that is, between users' workspaces), AccuRev organizes the data transfer into two steps:

1. One user makes his changes public — available to all the other members of his team. This step is called promotion.
2. Whenever they wish, other team members incorporate the public changes into their own workspaces. This step is called updating.



The first step involves a public data area, called a stream. AccuRev has several kinds of streams; the kind that we're discussing here is called a backing stream. The data in this public stream “is in back of” or “provides a backstop for” all the private workspaces of the team members.

AccuRev also allows you to save any number of intermediate versions of a file in your workspace, before making your changes public. Such “private” versions of a file are created by the keep operation.

## Establishing Your Identity

All AccuRev commands must be executed by a user who is listed in the AccuRev user registry. By default, the Integration uses your operating-system username as your AccuRev username. To have the Integration use a different AccuRev username, set environment variable `ACCUREV_PRINCIPAL` before starting the IDE. Here's a Bash Shell example:

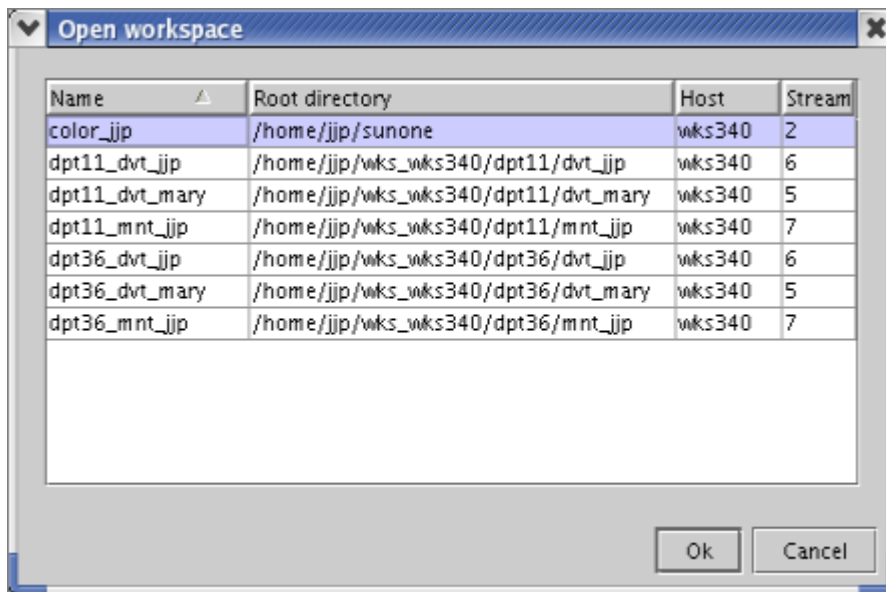
```
export ACCUREV_PRINCIPAL=derekp
./runide.sh
```

## Using a Workspace

The various Integration commands move data between the central source-code repository (called a depot) and your personal work area (called a workspace). You must work in an existing AccuRev workspace. You must create this workspace with the AccuRev GUI or CLI; there is no Integration command to create a workspace.

To work with the files in a particular workspace, you must “mount” it. (This is the IDE's terminology; it's different from the basic Unix/Linux concept of making a disk partition available

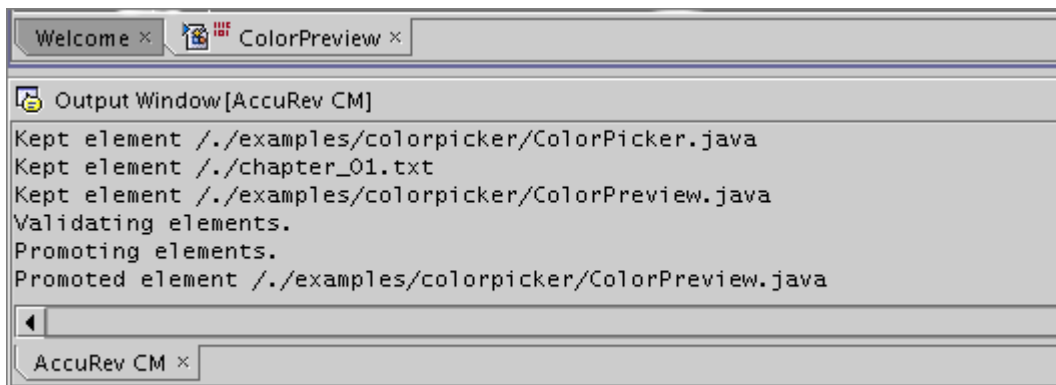
as a file system.) The command **Versioning > Mount AccuRev workspace** displays a list of all workspaces:



To mount a workspace in the IDE, select it and click **Ok**.

## AccuRev Command Output

AccuRev commands generate confirmation messages, which are displayed within the IDE's Output Window pane, in a tab labeled **AccuRev CM**.

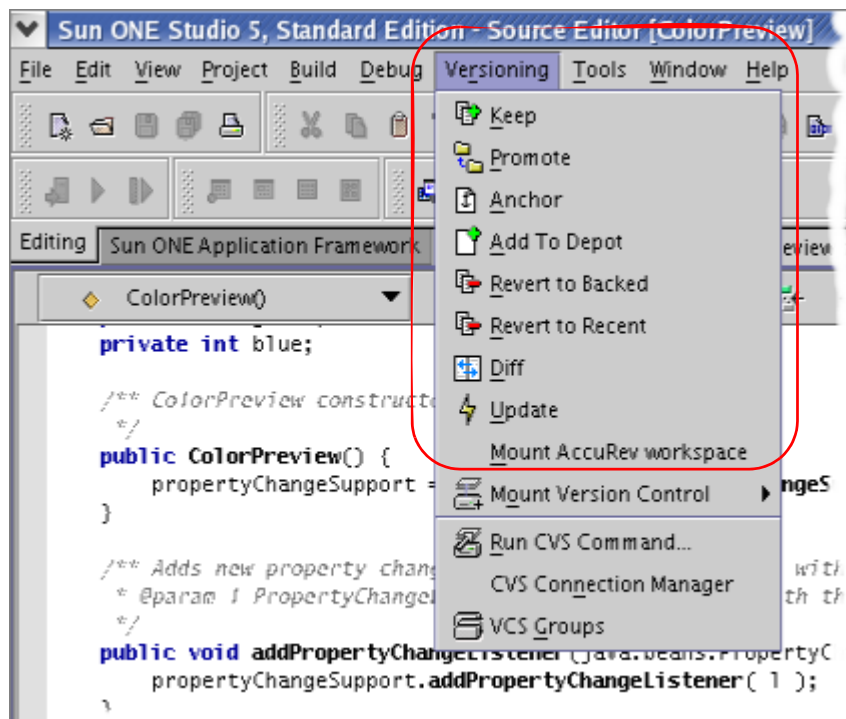


## Command Summary

All AccuRev commands supported by the Integration appear in the IDE's **Versioning** menu. These commands operate on the file currently displayed in the IDE's Editing pane.

The same set of AccuRev commands also appear in the context menu (right-click) for files in the Filesystem Explorer. These commands operate on the currently selected file(s) in the Explorer pane.

The AccuRev commands supported by the Integration are described in the sections below.



### Keep

The **Keep** command saves the changes you've made to one or more files as "private" versions in the AccuRev repository. These versions are visible only in your workspace — not in the "public" backing stream or in other users' workspaces.

### Promote

The **Promote** command converts one or more "private" versions into "public" versions. That is, it takes versions that you previously created in your workspace with **Keep**, and sends them to the backing stream shared by you and other members of your development team.

### Anchor

Many version control systems have a "check out" command that makes a file writable, so that you can edit it. But with AccuRev, your files are *always* writable. AccuRev *does* have its own similar command, called **Anchor**. This doesn't change the writability of a file, but it does activate the file in your workspace (that is, adds it to the workspace's default group). One effect of this is to ensure that the file won't be overwritten by an **Update** command. Under normal circumstances, you rarely need to invoke the **Anchor** command.



## Add to Depot

The **Add to Depot** command converts one or more of the files in the development project into AccuRev version-controlled elements. (The directory containing the files is also converted to an element, if necessary.) The files to be converted must be located in a directory within an existing AccuRev workspace.

## Revert to Backed

The **Revert to Backed** command discards the changes you've made to a file. It restores the version that was in the backing stream at the time of your most recent **Update**. (But if you promoted one or more versions of the element to the backing stream since your most recent **Update**, it restores the most recently promoted version.)

## Revert to Recent

The **Revert to Recent** command is similar to **Revert to Backed**, but rolls back a file only as far as the private version you recently created with **Keep**. This command is useful when you edit a file, save it with **File > Save**, then decide to throw away the changes.

## Diff

The **Diff** command compares your file with the private version you recently created with **Keep**. ("What have I changed since my last **Keep**?")

Note: only text files can be compared.

## Update

The **Update** command copies versions from your workspace's backing stream into your workspace. This has the effect of incorporating other people's changes, which they have promoted to the backing stream, into your workspace.

```

private static final String nl = "\n";
private static final String lineSeparator = "line-separator";

public static final String viewClass(Class c)
    throws Exception
{
    return viewClass(Class.forName(c.getName()));
}

public static final String viewClass(Class c, Object obj)
    throws Exception
{
    return viewClass(obj.getClass());
}

public static String viewClass(Class c)
    throws Exception
{
    if (c == null) {
        return "Error: null reference";
    }

    StringBuffer out = new StringBuffer(1024);
    out.append(Modifier.toString(c.getModifiers()));
    out.append(" class ");
    out.append(c.getSimpleName());
    out.append(nl);

    out.append(" extends ");
    out.append(ClassViewer.viewClass(c.getSuperclass()));
    out.append(nl);

    Class info[] i = c.getInterfaces();

    for (int i = 0; i < i.length; i++)
        out.append(ClassViewer.viewClass(i));

    out.append(nl);

    Constructor[] constructors =
        c.getConstructors();

    for (int i = 0; i < constructors.length; i++)
        out.append(ClassViewer.viewConstructor(constructors[i]));

    out.append(nl);
    out.append(nl);
}

```

# AccuRev

AccuRev, Inc.  
10 Maguire Road  
Lexington, MA 02421

Telephone: 781-861-8700  
E-mail: [support@accurev.com](mailto:support@accurev.com)  
Website: [www.accurev.com](http://www.accurev.com)